

University of Warwick institutional repository: <http://go.warwick.ac.uk/wrap>

A Thesis Submitted for the Degree of PhD at the University of Warwick

<http://go.warwick.ac.uk/wrap/59461>

This thesis is made available online and is protected by original copyright.

Please scroll down to view the document itself.

Please refer to the repository record for this item for information to help you to cite it. Our policy information is available from the repository home page.

Representing Intelligent Decision Making in Discrete Event Simulation: A Stochastic Neural Network Approach

by

Stephen Curram

MSc (Warwick)

BSc (Kent)

A thesis submitted in partial fulfilment of the requirements for the degree of
“Doctor of Philosophy”



University of Warwick

Warwick Business School

January 1997

Contents

Chapter 1

Introduction 1

1.1	Background	1
1.2	Method of Approach	3
1.3	Overview of Chapters	4

Chapter 2

A Review of Approaches for Representing Decision Making Behaviour in Simulation Models 6

2.1	Simulation Modelling	6
2.1.1	Simulation Modelling Approaches	6
2.1.2	Introduction to Discrete Event Simulation	7
2.1.3	Discrete Event Simulation Modelling Approaches	9
2.1.4	Comparison Between the Simulation Approaches	12
2.2	Simulation Objects	12
2.3	Intelligent Agents	14
2.3.1	Intelligent Decision Making and Intelligent Agents	14
2.3.2	Aims in Representing Intelligence	15
2.3.3	Decision Making in the Simulation Context	17
2.3.4	Paradigms for Representing Intelligence	20
2.3.5	Implications of AI Paradigms for Intelligent Agents in Simulation	27
2.4	Desirable Features of Approaches to Representing Decision Making	28
2.4.1	Accuracy of Representation	28
2.4.2	Ability to Represent Variety in Decisions	29
2.4.3	Data Driven Model Development	29
2.4.4	Clarity of Representation	30
2.4.5	Explanation of Decision Outcome	30
2.4.6	Allowance for Uncertainty	30

2.5	Current Representations of Intelligent Decision Making	30
2.5.1	Abstraction	31
2.5.2	Knowledge-Based Systems	32
	2.5.2.1 Defining a Knowledge-Based System	32
	2.5.2.2 Combining Knowledge-Based Systems with Simulation	34
2.5.3	Rule Induction	41
2.5.4	Fuzzy Sets	43
2.6	Alternative Representations of Decision Making	45
2.6.1	Case-Based Reasoning	45
2.6.2	Neural Networks	46
2.7	Evaluating the Approaches	49
2.7.1	Abstraction	49
2.7.2	Knowledge-Based Systems	50
2.7.3	Rule Induction	53
2.7.4	Fuzzy Sets	55
2.7.5	Case-Based Reasoning	56
2.7.6	Neural Networks	58
2.8	Comparison of Approaches	60
2.8.1	Summary of Strengths and Weaknesses of Approaches	60
2.8.2	Potential for Further Investigation	63

Chapter 3

	The Practical Use of Neural Networks with a View to the Representation of Intelligent Decision Making	64
3.1	The Principles of Neural Networks	65
3.1.1	The Origins of Artificial Neural Networks	65
3.1.2	The Structure of a Neural Network	67
3.1.3	Neural Network Training Paradigms	70
3.2	The Multilayer Perceptron	70
3.2.1	Features of the Multilayer Perceptron	70
3.2.2	The Functional Form of the Network	71
3.2.3	The Multilayer Perceptron as a Statistical Tool	77

3.3	The Practice of Using the Multilayer Perceptron	79
3.3.1	Selecting Variables	79
3.3.2	Coding the Variables	80
3.3.3	Setting Gain and Momentum Parameters	82
3.3.4	Generalisability of Results	83
3.3.5	Analysing Neural Network Results	88
3.4	Neural Network Representations of Stochastic Processes	91
3.4.1	The Characteristics of Stochastic Neural Network Representations	91
3.4.2	Real-Valued Decision Outcomes	92
3.4.3	Classification Type Decision Outcomes	94
3.4.4	Overview of Stochastic Representation	97

Chapter 4

Examining Stochastic Neural Network Representations using Artificially Generated Data

4.1	Method of Investigation	99
4.2	The Development of the Neural Network Software	100
4.3	Representing Continuous Valued Output Variables	101
4.3.1	Approach to Representing Decisions with Continuous Valued Outputs	101
4.3.2	Continuous Data Sets	105
	4.3.2.1 Beta Distribution Varying α_1	106
	4.3.2.2 Beta Distribution Varying α_1 and α_2	108
4.3.3	Neural Network Training and Testing	109
4.3.4	Results of Experimentation	110
	4.3.4.1 Beta($\alpha_1, 1$) Distribution	110
	4.3.4.2 Beta(α_1, α_2) Distribution	114
4.3.5	Investigating the Effects of Class Size	118
4.3.6	Overview of Results for Continuous Valued Output Models	119

4.4	Representing Classification Outputs	120
4.4.1	Approaches to Representing Decisions with Classification Outputs	120
4.4.1.1	Training with a Probability Input Variable (INVAR)	120
4.4.1.2	Training with Class Membership Probabilities as Outputs (OUTDATA)	121
4.4.1.3	Allowing the Network to Determine Class Probabilities (OUTNET)	125
4.4.2	Discrete Data Sets	125
4.4.2.1	Binomial Data Sets using Random Variable as an Input	126
4.4.2.2	Binomial Data Sets with Class Membership Probabilities as Outputs	127
4.4.2.3	Binomial Data with Network Attributing Class Membership Probabilities	128
4.4.3	Neural Network Training and Testing	128
4.4.4	Results of Experimentation	130
4.4.4.1	Results using Random Variable as an Input	130
4.4.4.2	Results using Class Membership Probabilities as Outputs	131
4.4.4.3	Results using Network Training to Attribute Probabilities	133
4.4.5	Investigating the Effects of Class Size	134
4.4.6	Comparison of Approaches for Modelling Classification Outputs	135
4.5	Independent Validation Data	137
4.6	Conclusions	138

Chapter 5

	Issues for the Practical Use of Stochastic Neural Network Models	141
5.1	Encoding the Situation	141
5.1.1	Specifying Variables	141
5.1.2	Specific vs General Specifications	143
5.1.3	Observed and Possible Behaviour	145
5.2	Hybrid Neural Network / Knowledge-Based System	146
5.2.1	Reasoning Behind a Hybrid Approach	146
5.2.2	Organisation of the Hybrid Model	147
5.2.3	The Task Allocation Unit	148
5.2.4	Neural Network Modules	148
5.2.5	Rule-Based Modules	149

Chapter 6

	Testing the Neural Network / Knowledge-Based Hybrid Model in Practice : The Bank Simulation	150
6.1	Choosing the Application	150
6.2	The Bank Simulation Model	152
6.2.1	Overview of the Model	152
6.2.2	Bank Activities	154
6.2.3	User Interactions	158
6.2.4	Specifying Scenarios	160
6.3	The Pilot Study	160
6.3.1	Aims of the Pilot	160
6.3.2	Data Collection	161
6.3.3	Modelling and Implementation	162
6.4	Data Collection	163
6.4.1	Considerations in Setting Up the Data Collection	163
6.4.2	Setting Up Scenarios	164
6.4.3	Specifying Exit Penalties	166
6.4.4	Recording Decision Data	167
	6.4.4.1 Arrival Decision Data	168
	6.4.4.2 In Queue Decision Data	170
6.4.5	Conducting the Data Collection Sessions	171
6.4.6	The Decision Making Questionnaire	173
6.5	Data Analysis and Model Building	175
6.5.1	Organising the Analysis	175
6.5.2	Identifying Sub-decisions and Dimension Reduction	177
6.5.3	Overview of Decision Upon Arrival at the Bank	178
6.5.4	‘Which Service ?’ Decision	179
6.5.5	Queuing Decision Upon Arrival at the Bank : ‘Join or Balk ?’	180
	6.5.5.1 Information : Join or Balk	180
	6.5.5.2 Transactions : Join or Balk	188
	6.5.5.3 Business : Join or Balk	195
	6.5.5.4 Currency : Join or Balk	198

6.5.6	Queue Joining Decisions	202
6.5.6.1	Transaction Queue Decision	202
6.5.6.2	Business Queue Decision	205
6.5.6.3	Currency Queue Decision	206
6.5.7	Reneging Decisions	207
6.5.7.1	The Decision Task	207
6.5.7.2	Issues for Building a Neural Network Model	209
6.5.7.3	The Network Training Data	214
6.5.7.4	Developing the Neural Network Model	216
6.5.7.5	Rule-Based Components	219
6.5.8	Queue Swapping Decisions	220
6.5.8.1	Queue Swapping Data	221
6.5.8.2	Rule-Based Decision Model	222
6.6	Implementing the Decision Modules	224
6.6.1	Implementation Strategy	224
6.6.1.1	Implementation for Special Customers	225
6.6.1.2	Implementation for All Customers	227
6.6.2	The Neural Networks	228
6.6.3	Arrival Decision Modules	229
6.6.4	Reneging Decision Module	230
6.6.5	Queue Swapping Modules	232
6.7	Observations on the Bank Simulation	233
6.7.1	The Bank Simulation as a Vehicle for Investigation	233
6.7.2	Decision Models	235
6.7.3	Decision Model Validation	236
6.7.4	Decision Model Implementation	238

Chapter 7

Discussion of the Stochastic Neural Network / Hybrid Modelling Approach	239
7.1 Research Context	239
7.2 Basis of the Approach	240
7.3 Results from the Artificial Data Sets	242
7.4 Results from the Bank Simulation Study	245
7.5 Strengths and Weaknesses of the Approach	251
7.5.1 Strengths	252
7.5.2 Weaknesses	254
7.5.3 Comparison with Other Approaches	255
7.6 Application in Practice	258

Chapter 8

Conclusions and Future Research	262
8.1 Conclusions	262
8.2 Future Research	265

References	268
-------------------	-----

Appendices

A : The Neural Network Software	A-1
A1 : Using the Neural Network Software	A-1
A2 : Neural Network Software Code	A-7
A2.1 Library Units	A-7
A2.2 Library Code Details	A-8
B : Bank Simulation Program	B-1
B1 : Program Structure	B-1
B2 : Input Distributions	B-2
B3 : Entity Attributes	B-3
B4 : MicroSim Commands	B-3
B5 : Bank Simulation Program Listing	B-23
C : Bank Simulation Scenarios	C-1

D : Bank Simulation - Participants Instructions	D-1
E : Example Decision Outcome Datasets	E-1
F : Bank Simulation Questionnaire	F-1
G : Decision Mapping Forms for Bank Simulation	G-1
H : Stay / Balk Decisions for the Bank Simulation	H-1
H1 : Information Service	H-1
H2 : Transactions Service	H-13
H3 : Business Service	H-24
H4 : Currency Service	H-28
I : Reneging Decision for the Bank Simulation	I-1
J : Implementation of Intelligence Modules in the Bank Simulation	J-1
J1 : Customer Attributes	J-1
J2 : Decision Module Code	J-2

Tables

2.1	Strengths and Weaknesses of Approaches to Representing Intelligent Decision Making	60
4.1	Input values for Beta($\alpha_1, 1$) : 66 examples (Small-Structured)	107
4.2	Ranges and Mid-Points for α_1 : 66 Examples (Small-Random)	107
4.3	Input values for Beta($\alpha_1, 1$) : 231 examples (Large-Structured)	107
4.4	Ranges and Mid-Points for α_1 : 231 Examples (Large-Random)	108
4.5	Beta($\alpha_1, 1$) : Mean Square Errors & Mean Absolute Deviations of the Neural Network Models	111
4.6	Beta(α_1, α_2) : Mean Square Errors & Mean Absolute Deviations of the Neural Network Models	115
4.7	Results for Beta(α_1, α_2) Large Data Set with Large Class Intervals	118
4.8	Summary of Results from Continuous Distribution	119
4.9	Input values for Binomial : 36 examples (Small-Structured)	127
4.10	Ranges and Mid-Points for p : 36 Examples (Small-Random)	127
4.11	Input values for Binomial : 121 examples (Large-Structured)	127
4.12	Ranges and Mid-Points for p : 121 Examples (Large-Random)	127
4.13	Results for Binomial with Random Variable as Input	130
4.14	Results for Binomial with Membership Probabilities as Outputs	132
4.15	Results for Binomial with Network Attributing Membership Probabilities	133
4.16	Results for Large Data Set with Large Class Intervals	134
4.17	Comparison of Approaches using Binomial Data	135
6.1	Summary of Scenarios for the Bank Simulation	165
6.2	Format of Data Collected for Arrival Decision	168
6.3	Format of Data Collected for In-Queue Decision	170
6.4	Information Data showing Percentage of Customers Staying	180
6.5	Transaction Data showing Percentage of Customers Staying	189
6.6	Business Data showing Percentage of Customers Staying	195
6.7	Currency Data showing Percentage of Customers Staying	198
6.8	Transaction Queue Choices	203
6.9	Business Queue Choices	205
6.10	Business Queue Choices for the Reduced Problem	206

6.11	Currency Queue Choices	206
6.12	Currency Queue Choices for the Reduced Problem	207
6.13	Summary of Reneging Cases	208
6.14	Proportion of Customers Reneging for each Service	208
6.15	Group Ranges for Reneging Decision Data	215
6.16	Extract from Reneging Data Set Showing Cumulative Probabilities	216
6.17	Summary of Decision Groups and Decision Models	219
6.18	Summary of the Number of Queue Swapping Cases	221
6.19	Analysis of Queue Swapping and Queue Swapping Opportunities	222
7.1	Continuous Data Results Showing Mean Absolute Deviations (MAD)	243
7.2	Discrete Data Results Showing Percentage of Correct Classifications	244

Figures

3.1	A Biological Neuron	67
3.2	A Typical Artificial Neuron	67
3.3	Typical Neuron Activation Functions	68
3.4	A Three Layered Neural Network Architecture	69
3.5	Illustration of a 3*2*3 Multilayer Perceptron	72
3.6	An Illustration of Underfitting, Overfitting and Generalisation	84
4.1	Example Neural Network Structure	105
4.2	Density Functions for the Beta(α_1, α_2) Distribution	106
4.3	Beta($\alpha_1, 1$): Residual Plots for Small-Structured Data Set (2*8*1)	112
4.4	Beta($\alpha_1, 1$): Residual Plots for Small-Random Data Set (2*1*1)	112
4.5	Beta($\alpha_1, 1$): Residual Plots for Large-Structured Data Set (2*8*1(L))	113
4.6	Beta($\alpha_1, 1$): Residual Plots for Large-Random Data Set (2*2*1)	113
4.7	Beta(α_1, α_2): Residuals for Small-Structured Data Set (3*12*1(L))	116
4.8	Beta(α_1, α_2): Residuals for Small-Random Data Set (3*8*1)	116
4.9	Beta(α_1, α_2): Residuals for Large-Structured Data Set (3*12*1(L))	117
4.10	Beta(α_1, α_2): Residuals for Large-Random Data Set (3*12*1)	117
4.11	Examples of Binomial(4,p) Distribution	126
4.12	Classifications for Best Networks with Random Variable as Input	131
4.13	Classifications for Best Networks with Class Probabilities as Outputs	132
4.14	Classifications for Best Networks Determining Class Probabilities	134
5.1	Screen Display of the Petrol Station Model	143
5.2	Neural Network / Knowledge-Based Hybrid	147
6.1	Screen Shot of Bank Simulation	153
6.2	Activity Cycle Diagram for Bank Customers	155
6.3	Activity Cycle Diagram for Human Controlled Customers	156
6.4	Service Distributions for Bank Simulation	157
6.5	User Selection Options for a Transaction Customer	159
6.6	Screen Shot of Bank Simulation Showing Exit Penalty	167
6.7	Overview of Decision upon Arrival	179
6.8	Information Test Results for the INVAR Network	183

6.9	Information Test Results for the OUTDATA Network	185
6.10	Information Test Results for the Bounded OUTDATA Network	186
6.11	Information Test Results for the OUTNET Network	187
6.12	Transaction Test Results for the OUTNET Network	193
6.13	Transaction Test Results for the Bounded INVAR Network	194
6.14	Business Tests Results for the INVAR Network	197
6.15	Currency Results for the INVAR Network	200
6.16	Currency Results for the Bounded INVAR Network	201
6.17	Currency Results for the Bounded OUTDATA Network	201
6.18	Renege Model Test Results for 2*6*2 Network	217
6.19	Screen Shot of Model with Decision Modules for Special Customers	226
6.20	Screen Shot of Model with Decision Modules for All Customers	228

Acknowledgements

A number of people have offered help and encouragement during the course of this research, to whom I am indebted.

Firstly I would like to thank Robert Hurrion and John Mingers for their time and extremely useful comments during the preparation of the thesis. Also thanks go to Adrian Escott for his proof reading, and to Zoë Evans for converting my neuron sketch to an electronic format and for her role as chief guinea pig for my experiments.

Also I wish to thank my friends for their encouragement and support during this research. In particular to Bernadette Baker (for saying the right things at the right times), Frances O'Brien, Susanna Hogan, Andrew Martin, Maureen Meadows, Carol Moxham and Iain Munro.

Finally, special thanks go to my parents for encouraging me in my exploits, and for always being there.

Abstract

The problem of representing decision making behaviour in discrete event simulation was investigated. Of particular interest was modelling variety in the decisions, where different people might make different decisions even where the same circumstances hold.

An initial investigation of existing and alternative approaches for representing decision making was carried out. This led to the suggestion of using a neural network to represent the decision making behaviour in the form of a multi-criteria probability distribution based on data of observed decision making.

The feasibility of the stochastic neural network approach was investigated. Models were fitted using artificial data from discrete and continuous distributions that included the shape parameters as inputs, and tested against known results from the distributions. Also a bank simulation was used to collect data from volunteers who controlled the queuing decisions of customers inside the bank. Models of their behaviour were created and implemented in the bank simulation to automate the decision making of customers.

The investigation established the feasibility of the approach, although it indicated the need for substantial amounts of data showing examples of decision making. A hybrid model that combined the stochastic neural network approach with a rule-based approach allowed the development of more general models of decision making behaviour.

[Keywords: Discrete Event Simulation, Neural Networks, Multilayer Perceptron, Artificial Intelligence, Behaviour, Decision Making, Intelligent Agents, Stochastic Processes]

Chapter 1

Introduction

1.1 Background

Discrete event simulation involves modelling the behaviour and interactions of entities in a system. Entities can, for example, represent customers in a service system, products in a manufacturing system or paperwork in an office. The models generally involve situations where activities can occur that take a specific amount of time, usually determined by sampling from a probability distribution, and where there are queues due to limited availability of resources. Entities are modelled by specifying their behaviour at each stage of the system. The behaviour of entities can usually be modelled simply since it exists within tightly bound limits. Any choices faced by entities are extremely limited and require only simple rules to model them. However, some situations exist where the behaviour of entities is not straightforward, and the choices made should be based on the states of a number of criteria that exist in the system. In other words, the entities are required to exhibit some intelligence. In these situations, the ordinary approaches in discrete event simulation are generally too limited to model the behaviour effectively.

In most cases intelligent behaviour is ignored or handled in a highly simplified manner. The simulations use either simple rules or random sampling to determine the outcome. These generally take little or no account of the decision criteria. In some cases more advanced knowledge-based systems are used that take account of the factors that affect the decision making, and model the causal relationships that lead to the final decision. The knowledge-based approach generally assumes that all of the entities will act in a similar manner. Given a particular set of circumstances, all the entities will follow the same decision making process and end up making the same decision.

An area that is usually ignored is modelling where people may actually make different choices given the same set of circumstances. This does not mean that they ignore the decision criteria, but that they differ in their reaction to the criteria. Usually either the

differences in the decision making are ignored, or a probability distribution is used and most of the decision criteria ignored.

This study involves looking at modelling approaches that allow the differences in the decision making behaviour of people to be modelled, at the same time as taking account of the factors that affect the decisions. This means that there is an interest in actual behaviour rather than some sort of optimal decision making. The purpose of modelling such a situation is to accept that in some cases people's innate decision making behaviour cannot be changed, but that aspects of the system could be designed so as to affect the decision criteria. For instance, it is difficult to change directly the behaviour that people exhibit when driving in certain circumstances, but the design of the traffic flow systems might change the circumstances and therefore indirectly change the behaviour.

The original brief of the study was broad, looking at modelling approaches in general. During the review of literature the study focused on neural networks as an approach that might offer a way of achieving the twin goals of representing stochastic variation in decision making while still taking account of the decision criteria. The approach was substantially different to any that had been used before, so the main purpose of the study became the evaluation of the neural network approach.

The aims of the study were :

1. To review the current approaches for representing intelligent decision making, with consideration of alternative approaches,
2. To develop a stochastic based approach for representing decision making behaviour using neural networks,
3. To evaluate the feasibility of the stochastic neural network approach in terms of its representational accuracy and the issues involving its practical application.

1.2 Method of Approach

The research was investigative by nature. It involved determining an area of research interest, identifying modelling goals, identifying and developing an approach to try to achieve those goals, and determining the feasibility of that approach.

The investigation required a careful step by step approach. Although simulation is an established area of Operational Research, and neural networks are currently receiving a lot of research attention, little work has been done in combining the areas together. The development of the new approach had a number of possible routes that could be followed. Thus care was taken to divide the investigation into a series of stages, and review the results before continuing onto the next stage. The aim was to make the decisions on the direction of research as informed as possible at the time.

The first stage of the investigation was to review the literature on the representation of intelligence in general, and more specifically in simulation. Literature was also reviewed for the application of the neural network technique. The review of literature continued throughout the study, particularly for the neural networks which is still a rapidly developing area. There is also a high experiential element to the effective use of neural networks which was gained by applying the approach to other problem areas.

The stochastic neural network models were developed on the theoretical grounds of the reported abilities of neural networks, and partly by considering their use on hypothetical problems. The first stage of testing out the models was their use on artificially generated data sets. This allowed the use of known functions against which the results of the model could be judged. The initial experiments were ad hoc, to see if there was any validity in the approach. Once this was confirmed, a structured experimental framework was designed for analysing the accuracy of models.

It was considered important to be able to ascertain some of the practical implications of using the stochastic neural networks in a simulation application. The choice of application was limited by the availability of data. It was known that any application

would be a specific case, so care was required in generalising the lessons from the experiment.

The limitations of any single study mean that not every issue surrounding the new approach can be addressed. The aim was to address the fundamental issues of the feasibility of the approach. The outcome of the study was bound to raise other issues for investigation.

1.3 Overview of Chapters

Chapter 2 reviews the area of representing intelligent decision making in simulation as it currently stands. There is a consideration of the paradigms for representing intelligence, and the requirements of an approach for the representation intelligent behaviour that allows variability in decision making. There is a review of methods and case studies that have been described in the literature, and possible alternative methods are identified. Finally there is a comparison of the methods with regard to the potential for further research.

Chapter 3 considers the structure and use of neural networks in detail. The basic principles are described, with particular concentration on the structure and use of one type of network, the multilayer perceptron. There is then consideration of the formulation of data structures that would allow decision criteria and stochastic elements to be represented by neural network models.

Chapter 4 describes experiments with the stochastic based neural networks using artificially generated data from known probability distributions. The results from the neural network models are compared against the known values for the distributions to evaluate the representational accuracy.

Chapter 5 considers issues and adaptations required for the neural network approach for it to be used in practical applications. A hybrid approach is suggested that combines neural network and rule-based components together to produce more general models of decision making.

Chapter 6 describes the use of the neural network approach on a practical case study involving simulating the queuing behaviour of customers in a bank. This is used to determine the issues involved in using the approach in practice in terms of specifying the decision making, training and validating the neural networks, and implementing the models in a simulation.

Chapter 7 draws together the issues raised in the previous chapters and discusses the conditions necessary for the neural network approach to be applied in practice. A number of potential application areas are proposed.

Finally, Chapter 8 highlights the main conclusions from the study and suggests areas for further research.

Chapter 2

A Review of Approaches for Representing Decision Making Behaviour in Simulation

This chapter reviews the area of representing decision making behaviour in simulation as it currently stands, and examines other approaches that could have potential benefits. Section 1 describes the main features of discrete event simulation. Section 2 discusses the use of object oriented approaches to simulation. In Section 3, a description of intelligent agents is given, with some discussion on the main paradigms for representing intelligence. Section 4 describes some desirable features for an approach to representing intelligent decision making. In Section 5, approaches to representing intelligence in simulations which are currently being used or have been suggested are described. Section 6 looks at some alternative approaches which might have potential. All of the approaches discussed are evaluated in Section 7. Finally in Section 8 there is a comparison of the methods with regard to potential for further research.

2.1 Simulation Modelling

2.1.1 Simulation Modelling Approaches

There are several forms of simulation which exhibit different features and different modelling methods. Static simulations represent a system at a particular point in time, these could be scale models of buildings, or Monte Carlo simulation used to evaluate mathematical functions. Continuous simulations involve modelling continuous changes in the state of a system with respect to time. Systems dynamics is an example of an approach for continuous simulation. It is very much interested in causal loops, rates of change and long term dynamics of a system. Discrete event simulation treats changes over time as a stepwise rather than a continuous process. In the simulation, events in the system are seen to occur at particular points in time. So time is advanced in steps, every time an event occurs.

Out of all the simulation approaches, discrete event simulation is the one which is generally most concerned with tracing the actions of individual objects. The models are very much based on queueing systems where the objects can be involved in an activity (performing some task, being processed etc.) or be in a waiting state (usually a queue) where they are waiting for the right conditions to start an activity.

Since the focus of the thesis is on representing intelligent decision making, it is most natural that this should involve models of individual decision makers. Thus discrete event simulation is the most natural approach in which this should take place. This does not necessary preclude the use of the models which are developed in other forms of simulation, and this issue will be considered further at the end of the thesis.

2.1.2 Introduction to Discrete Event Simulation

Thesen and Travis (1992) describe simulation as "the use of a computer model to mimic the behaviour of a complicated system and thereby gain insight into the performance of that system under a variety of circumstances".

Essentially the simulation model is a dynamic technique for representing a dynamic system. The dynamic nature of the system means that the components of the system change over time. For instance, a bank system may be defined by the customers in a particular branch who wish to avail themselves of its services. An individual customer at various points in time may be entering the bank, queuing at a service counter, being served at a particular counter. At different points in time the system will be in a particular *state* that is represented by the activities in which various components of the system are involved.

The form of simulation which will be referred to from now on is *discrete event simulation*. This is where the state of the system is viewed as changing in distinct steps, as opposed to a continuous gradual change. Thus a customer in the bank changes from being in a queue to being served, and then from being served to having finished being served in discrete jumps. This is known as *next-event time advance* where the simulation model is only updated when a new event in the system causes its

state to change. The simulation employs a time clock to measure when these discrete changes in state take place. A system which changes continuously over time can be approximated using the discrete event approach using *fixed-increment time advance*, where the simulation clock is advanced in fixed steps and the state of the system is updated at each step. The fixed-increment time advanced model could be used to represent situations such as cars travelling down a stretch of motorway.

The use of the simulation model allows changes to be observed which may otherwise be too disruptive, expensive, time consuming or simply dangerous to try with the real system. The simulation may be used for quantitative experimentation to numerically compare the performance of a number of different system configurations. Also a visual-interactive simulation model as developed by Hurion (1980) can allow the user to learn more about a system through being able to make operational changes in the model and watch the results on a graphical representation of the system.

At this point it is useful to define what is meant by a *system*, at least from a discrete event simulation perspective. Schmidt and Taylor (1970) define a system as being a "collection of entities which act and interact together towards the accomplishment of some logical end". Entities include the 'customers' of the system who may be actual customers in a service system or the products in a manufacturing system, and the resources which perform some sort of processing of the customers and may be pieces of machinery or servers.

In order to create a simulation model of the system, the main entities must be identified and the behaviour of the entities must be understood and replicated, incorporating any random variations that may occur. Each entity may be involved in a number of activities, where the behaviour of the entity is represented by the choice, order and duration of activities, and how it interacts with the other entities in the system.

In many cases the behaviour of the entity is well understood because the system is relatively predictable. A factory production line, for instance, has a well-ordered set of processes where the behaviour of machinery and workers is laid down and tightly

controlled. Where variation in the time to perform an activity occurs, this can be modelled using a probability distribution.

Even in cases where alternative behaviour is possible, it may be modelled using probability distributions. For example, at a petrol station, 40% of customers require unleaded fuel, 50% four star, and 10% diesel. Using the probability distribution, each car arriving is allocated into one of the classes, and this class effects its behaviour in predictable ways at various parts of the system (which pumps it can go to, filling times etc.). When it comes to paying for the fuel, another distribution is used to find which method of payment is being used, and time to perform the activity allocated as appropriate.

Cases exist where the decision is more complicated, and is dependent on a number of highly variable but critical factors which are present within the simulation. If a clear policy exists in the system then this decision may already have been made (perhaps in the case of a machine breakdown), and so the behaviour is predictable and fairly easy to model. Where no clear policy exists, modelling the behaviour of the entity requires some element of representing intelligent decision making.

2.1.3 Discrete Event Simulation Modelling Approaches

Pidd (1992) describes four main approaches to building discrete event simulation models: Event-based, Activity-based, 3-Phase, and Process-based. All the methods are based on the principle that the simulation can only change state after an unconditional event occurs (i.e. one scheduled to occur after a certain amount of time) which may then allow conditional events to occur (events which happen if the correct conditions exist). Unconditional delays usually represent an entity being involved in an activity for a certain period of time, while conditional delays are usually queues. Each method uses an executive which records a list of the scheduled, unconditional events and advances the simulation clock onto the next event time. The differences between the approaches occur in the way that the execution of events is handled at a particular stage in the simulation, involving the resolution of unconditional and conditional events.

Event-based Approach

In the Event-based approach, the executive stores a list of unconditional events along with the time that each is due to occur. The occurrence of the next unconditional event causes the executive to start a segment of the simulation code. This segment has sole responsibility for resolving the particular unconditional event, as well as any conditional events that may be enabled by the change in state.

The advantage of the approach is that execution of the code is efficient, only dealing with those possibilities which may occur after a particular unconditional event. The disadvantage is that the code can become complicated since all possible consequences from a particular event must be handled by the specific segment. This can increase the difficulty in building and making changes to the model.

Activity-based Approach

The Activity-based approach has a simple executive that stores a list of times for unconditional events, but does not associate a particular event with those times. At a particular time, all possible activities are tested, one at a time. The activities have a condition to test if they should be used, either involving testing a time for an activity to see if an unconditional delay has finished, or circumstances for a conditional delay to see if it has been resolved. Each activity has code to represent its behaviour if the conditions are passed.

The activity-based approach has relatively simple activity code. Since each activity is considered individually, it is only necessary to consider one activity at a time, along with the links to neighbouring activities. This makes for simple, modular code that can be changed with relative ease. Care must be taken in defining the boundaries of an activity and the interactions with other activities, as well as ensuring that the conditions for the activity are accurate and exhaustive. The main drawback of the approach is that each event requires a check of the conditions of every activity making the approach inefficient. In addition, the order in which the activities are checked is important for the proper operation of the simulation. Those activities which have

knock on effects on other activities must be checked first so that the consequences of an event can be traced through the system.

3-Phase Method

The 3-Phase method can be thought of as a mix between the event-based and activity-based approaches as described above. The 3-Phase method (A,B and C phases) uses an executive like that of the event-based approach, keeping a list of the times of unconstrained delays along with the corresponding events. The A-Phase is the update of the clock to the next scheduled event. At the time that an event occurs, the executive starts the B-Phase which is a segment of code that resolves only the scheduled event. Thus the B-Phase code is made up of only those events which will occur after unconstrained delay, and only one of those events will be used at a particular time. When the B-Phase has been resolved, the C-Phase is started, which is similar to the activity-based approach. Here, only events which can occur after conditional delays are checked, to see if the conditions are such that the event can occur. As in the activity-based approach, the order that the conditional events are checked is important.

The 3-Phase method maintains much of the efficiency of the event-based approach, since only one unconditional event is checked for each clock period, but splits up events into manageable blocks as in the activity-based approach. This means that the simulation can be designed using simple blocks, while having relatively efficient execution times.

Process-based Approach

The Process-based approach uses the concept of entities having a pre-defined set of processes that they must pass through. At any particular point in time an entity (if in the system) will be involved in a constrained or unconstrained delay. The approach differs from the previous three in that it is based around entities, rather than around events. The executive keeps data on entities, recording for each the reactivation time (if involved in an unconstrained delay) and the reactivation event. It sorts entities into two lists: the future list that contains entities which will be reactivated after an

unconstrained delay, and the current list that has entities which can proceed if the conditions allow. The clock is advanced to the nearest time on the future list and the corresponding entity activated, followed by a check on all entities on the current list.

The process-based approach has an intuitive feel in that the analyst can map through the lifecycle of the entity from beginning to end. The approach is reasonably simple if the system has uncomplicated process paths. Interaction with other entities adds to the complexity, as does having various process paths for a particular entity.

2.1.4 Comparison Between the Simulation Approaches

The activity-based approach has the clearest and simplest programming approach, since each activity can be dealt with individually, however the need to check all conditions reduces its computational efficiency. The event-based approach has computational efficiency, in that only relevant events are tested, but at the expense of the ease of programming. The 3-phase method is a good compromise between the two approaches, having the programming simplicity of the activity-based approach but better computational efficiency. The process-based approach has an intuitive entity-orientated approach, but this is at the expense of programming ease, particularly as the process routes become more complicated.

Models that have elements of intelligent decision making are likely to have fairly complicated process routes, since the decision will often involve a choice of different routes. Therefore, in this analysis the 3-phase method is preferred. However, the basic approaches to representing intelligent agents should be valid for all four modelling approaches, the differences being with the final implementation details in the model.

2.2 Simulation Objects

While this thesis does not directly involve the use of object oriented simulation techniques, it is useful to be aware of the possibility of using objects to represent components of the simulation, including those that are required to make decisions. The following looks at approaches which have been suggested for modular simulation

models, such that they may allow 'intelligence' modules to be easily incorporated, regardless of the approach used to actually represent the intelligence.

Birtwistle et al. (1980) describe the concept of using objects as first introduced by SIMULA, a programming language designed for process-based simulation. An object is a simulation component made up of three parts: a header identifying the object, data structures that specify state variables, and an ordered sequence of actions to be carried out. The object allows for the classic process-based approach to be used. The data structures contain a list of the actions to be performed to allow the progress of the object to be marked. The action descriptions are a set of instructions for the behaviour of the object at each stage of the process. While this approach allows an object to contain its own behaviour, the limitations of the language do not allow for the easy insertion of 'intelligent' modules.

Joines et al. (1992) describe a C++ based system for process-based simulation. This allows simulation objects which exhibit the object-oriented features of utility of classes, encapsulation and class inheritance that allow objects to be re-used and adapted. The flexibility offered by this approach makes the concept of the slot in 'intelligence' modules more feasible. The constraints of the system are imposed by the process-based handling of events, which as discussed in Section 2.1.2, do not allow for the easy implementation of highly variable logic paths, as may be required for a decision making object.

Pidd (1995) is critical of the process-based approaches, and identifies weaknesses of the entity focused approach in complex systems where a high degree of interaction occurs between entities. The approach assumes that the executive can cycle through entities determining whether each can progress further through their process, however where interaction is concerned, independence is lost and the logic of the system can vary depending on the order in which entities are reviewed. Pidd suggests the use of the three-phase approach, being activity rather than entity focused. Here both the entities and activities are implemented as objects. The entities contain information about their own state, and some of their own specialised behaviour, while the activities

represent the overall behaviour in the simulation. Thus it is the activity which interacts with its client entities and specifies how each will be processed.

Overall, it is the three-phase method that looks most promising for handling the complex logic routing which is likely to result from using decision making units. A decision making module could be treated as an activity, specifying how the entity will act. However, a more attractive approach would be for the entity to contain its own decision making modules which interact with the appropriate activities to make a decision. This approach means that different classes can have different decision making mechanisms, as well as allowing decision methods to be developed and slotted into the appropriate entities.

2.3 Intelligent Agents

In the previous sections there has been some mention of incorporating intelligent decision making into a simulation. This section provides a more detailed discussion of what is meant by an intelligent agent, and the actual forms of decision making that would be represented in a simulation. In any discussion of intelligence, there must be consideration of the main paradigms for representing intelligence, a subject which is under fierce debate.

2.3.1 Intelligent Decision Making and Intelligent Agents

For the sake of clarity it is necessary to distinguish between the decision making that goes on in the real world, and the decision making in the simulated world. Despite the simulation being a representation of the real world, and the decision making in the simulation being a representation of real world decision making, the simulated world is bound to be a more abstract one, which is bound to influence the way that the decision making is represented. To this end, in the real world there will be what is termed an *intelligent decision maker*, while in the simulation there will be an *intelligent agent*.

An intelligent decision maker will usually be a human being but may also include (real world) 'intelligent' machines. The decision maker will be called upon to make decisions on what actions they or others will take. This requires that the decision maker has

some awareness of the conditions that exist that may effect the decision, and can apply their knowledge to come up with an appropriate action.

An intelligent agent is defined by Turban (1992) as an “Expert or knowledge-based system that is embedded in computer-based information systems (or their components) to make them smarter”. There is actually no reason why the intelligent agent needs to be a knowledge-based system rather than any other artificial intelligence approach. Within the simulation context, the aim is for the intelligent agent to make decisions in the simulation in a way that is representative of the decision maker in the real system, that is, to make the same sorts of decisions in the same circumstances.

Robertson (1986) offers another definition of intelligent agents, geared more directly for the simulation technique:

"[an] intelligent agent ... has a number of expert rules that support its behaviour and an 'agenda' that defines its goals. The intelligent agent then 'lives' in the simulated world until the 'agenda' is exhausted."

In Robertson's view "every object is an Intelligent object". The intelligent agent is given a number of goals to achieve, and needs to make decisions using rules on how to execute them. Most of the rules will be fairly simple and well-defined processing steps, particularly in the case of a document or a piece of machinery. Some goals may need more complicated rules for making decisions.

Robertson's definition of an intelligent agent is more general than the one used in this thesis. Here an intelligent agent is an entity in a model or simulation which is called upon to make a decision that requires weighing up alternatives based on a number of criteria, and where no simple policy exists for making that decision. No assumption or stipulation is made for the method by which the behaviour is represented.

2.3.2 Aims in Representing Intelligence

When talking about intelligent decision making or intelligent agents, it is useful to consider what is meant by intelligence. Jackson (1974, p5) considers the nature of intelligence summarising it as “the ability ‘to act rightly in a given situations’”. Jackson

notes the problems in coming up with an accepted and workable definition of intelligence. Sheil (1987) notes that “describing anything as intelligent means we don’t fully understand it” and adds “lacking any precise definition of what it means to be ‘intelligent’, most people will conclude that an intelligent computer system will behave much as a person would”.

In terms of creating artificial intelligent systems, Jackson (1974) and Sheil (1987) agree that it is not realistic to try to copy the details of human intelligence, but better to try to simulate the outward appearance of intelligence within a limited domain. Simon (1995) states “there is no immediately obvious reason why there should be any close connection between research directed at designing intelligent systems and simulating human cognition, or between the corresponding theories of intelligence”. However, Simon does later acknowledge that the two areas can learn from each other. The goal of only giving the outward appearance of intelligence, is often referred to as ‘soft AI’.

Others would see Artificial Intelligence, as Boden (1990) puts it, as “the science of intelligence in general”. The goal is to explain the way that intelligence works using models, and possibly to replicate human intelligence, modelling the internal workings rather than just the external appearance.

Standard simulation involves modelling processes to the appropriate level of detail. Representing the outward appearance of individual processes, and the interactions between processes is generally considered more important than the detail of how, say, a particular machine works. In a similar vein, in developing intelligent agents for simulation modelling the interest is in the less ambitious but more achievable aim of producing something that can give the appearance of being smart, without actually trying to model all the human processes that lead to intelligence. To do this, it is necessary to limit the intelligence to a very narrow domain. In the case of the intelligent agents in simulation, the intelligence involves making a decision about an action to take, based on certain criteria which can be observed within the simulation.

2.3.3 Decision Making in the Simulation Context

Simulation modelling is used for evaluating the performance of systems under differing configurations. This is done by modelling the behaviour and interactions of individual components in the system. This includes modelling the decision making behaviour of components of the system if this is thought likely to affect the operation of the system in any significant way.

It is worth considering what form a decision can actually take within the limited structure of a simulation model. One form of decision making is where there is a choice from one of a set of mutually exclusive possibilities. This can be thought of as choosing from a discrete set. For instance, in driving a car, there may be a decision at some point of whether or not to overtake the car in front. The set of possible decisions might be i) to overtake, ii) to not overtake yet. Another example, is where a customer entity in a supermarket needs to decide which queue they will join for a checkout counter (one out of all the possible counters).

Another form of decision is where there needs to be control of some continuous process. The decision is the amount of change which is to be made to some parameter. Again using the example of driving a car, there might be the need to decide whether to change the acceleration (or deceleration) of the car, and by how much, in response to circumstances (speed limits, other cars etc.). In both cases the decision is likely to involve the consideration of a number of factors, some of which the intelligent agent may not be accurately aware of due to limits in what can realistically be perceived by the decision maker in real life.

In an abstract sense, only these two forms of representing decisions are necessary, since all other decision making can be formulated as one of these two. For instance, in the discrete case, if there is a set of decisions from which several can be chosen at once, it is possible to define this as a set of mutually exclusive decisions.

Modelling this decision making behaviour will generally be more difficult than modelling the behaviour of ordinary entities, and cannot generally be handled by the

(relatively) simple rules and probability distributions that are usually employed in simulations.

In order to consider the types of decisions that might be made within the simulation context, the notion of three archetypes for decision makers (and their equivalent intelligent agents) has been developed, these are the *Machine Artificial Intelligence*, the *Expert or Controller*, and the *Customer*.

Machine Artificial Intelligence

Perhaps the simplest type of intelligent decision maker, in terms of understandability, is the Machine Artificial Intelligence (MAI), such as an automatic guided vehicle. This machine will generally have clear and transparent criteria for making decisions, and should make consistent decisions (i.e. always make the same decision given a particular situation).

The system for representing intelligence should already be in existence for the MAI, and provided it is software based, and can be interfaced with the simulation software, it can be used directly. In terms of representing an existing MAI, the task is relatively trivial, and so will not be considered further in the main investigation of the thesis. However, insights from developing a system to represent intelligent decision making in simulations may help in setting up an MAI in the first place.

Expert (or Controller)

The (Expert) decision maker is a human being who, in some way, has control of the whole or part of the system. This will often be a manager or operator of some process. This person is likely to have good knowledge of the system and can often be thought of as an 'expert' in that system. Whether or not that person makes optimal decisions, and whether the simulation should attempt to replicate that decision making or attempt to improve on it (and thus aid better decision making), is dependent on the requirements of the study.

Being closely related to the system of interest, the person should be available for discussions on their decision making. That person might be expected to be reasonably consistent in their decisions, but not perfectly so.

The review of the literature later in the chapter shows that in cases where there is some effort put in to representing intelligent decision making in a simulation, it is most commonly the Expert who is modelled.

Customer

The Customer of a system may also be called upon to make intelligent decisions in deciding on how to act within the system. This is most likely to occur in a service type operation. The Customer is unlikely to have intimate knowledge of the full workings of the system, but may base their decisions on 'common sense' knowledge developed outside or based on partial (customer perceived) knowledge of the system. This means that while the Customers may not be seen to be employing 'expert' skills, they are likely to be using skills of every day decision making.

The decisions that the Customer makes may not be optimal (as far as the system or the Customer is concerned), being based on limited experience with the system. A Customer's interaction with the system is likely to be as part of a wider system (outside the scope of the simulation) that can lead them to bring other criteria which are not observable into the decision process. Due to these extra criteria, different perceptions of the situation, and differences in evaluating uncertainty, it is likely that different Customers will make different decisions in the same situation. This means that a system which represents the decision making of different Customers is likely to have to represent different decisions even when the decision criteria are the same.

There is also an issue of acquiring information for how Customers make their decisions. Being temporary visitors to the system, they may not be available for discussion of their decision making.

The detailed representation of customer decision making is largely ignored in simulation modelling. While there are particular difficulties in representing the behaviour of customers, there are situations where this can have a significant impact on the performance of the system. Indeed, it is the previous lack of interest which makes this type of decision maker of particular interest to this study.

Combining Archetypes

A mixture of archetypes is possible, particularly for Experts and Customers. For instance, a car driver may be seen as having intimate knowledge of the system (car, road layout, highway code, etc.) and so may be seen as an Expert, although their decisions may not be optimal in terms of safety or traffic flow (typical aims for a traffic system as a whole). Car drivers are likely to have a limited perception of the traffic flow as a whole, bring in decision criteria from outside the traffic system, and display different decisions in similar situations, so that they display characteristics of the Customer agent. Car drivers may also not be available to try and explain their decision making.

2.3.4 Paradigms for Representing Intelligence

Boden (1995) suggests that most AI scientists fall into one of two computational paradigms, which Boden calls *classical AI* and *connectionism*, although other paradigms do exist.

Classical AI is a symbolic approach where knowledge and expertise is represented through the explicit definition of concepts and relationships. These sorts of structures are often called cognitive or rationalist. The symbolic approach has the advantages of being explicit, and hierarchical. Opponents of the approach claim that human intelligence is not just based around the manipulation of symbols. It involves intuition, something that is said to develop through experience and goes beyond rational rules.

The connectionist, and in particular Parallel Distributed Processing (PDP) approach involves trying to develop brain-like structures, representing knowledge as patterns of activity rather than explicit symbols. This lack of explicitness means that the

relationships and knowledge that are being captured are not clear. However, the PDP models are able to learn from example, rather than being programmed as with the classical AI approach. One of the main criticisms of the PDP approach is that the structures which are being modelled are extremely simple compared to the human brain, and in many cases their working shows little resemblance to the biological counterpart.

Other paradigms do exist. These often come under the heading of embodied entities, involving a concentration on behaviour that is bound up with, rather than separated from, the real world environment. From these, Boden (1994) notes the approach of situated robots that interact with the world through hardwired behaviours with competition between the low level behaviours, and some higher level (but non-programmed) interactions.

It is worth looking in more detail at the arguments which have been made for and against the paradigms, and see how they relate to the problem in hand of representing decision making in simulations.

Classical AI

Since the classical paradigm is the most established, it will be used as the starting point for discussion. A number of terms are used which fall into this paradigm : cognitivism, rationalistic, symbolic and representational. Although some would argue that there are differences between some of the concepts represented by the terms, they will be taken together as being indicative of the classical paradigm.

Winograd & Flores (1986) describe the discipline of cognitive science as coming from within the rationalistic tradition. This tradition is depicted by Winograd & Flores in the following steps :

1. Characterise the situation in terms of identifiable objects with well-defined properties.
2. Find general rules that apply to situations in terms of those objects and properties.

3. Apply the rules logically to the situation of concern, drawing conclusions about what should be done.

Norman (1981) stresses the importance of symbolic representations in cognitive science, saying “to some, the very essence of a cognitive system is that of a symbol processing system” and “issues of representation and processing form the basis of Cognitive Science”.

Newell (1981) draws comparisons between computer technology and human beings as symbol processors. Computers have various levels : *device level* involves the physical electronics, *circuit level* consists of the electric currents, *logic level* involves the processing of bits, while the *program level* contains the data structures, symbols and programs, and on top of that is the *Processor-Memory-Switch level* which is the medium that contains data and information. Newell states that “the logic level structure that creates a particular symbol system is called the *architecture*”. Newell describes humans as “physical symbols systems”, so that “there must exist a neural organisation that is an architecture -i.e., that supports a symbol structure”.

Varela (1992) states that for cognitivism, “the central intuition is that intelligence (including human intelligence) so resembles a computer in its essential characteristics that cognition can be *defined* as computations of symbolic representations”. Varela also summarises the cognitivist research programme using the set of questions and answers which are recounted below :

Question # 1: What is cognition:

Answer: Information processing: Rule-based manipulation of symbols.

Question # 2: How does it work?

Answer: Through any device which can support and manipulate discrete physical elements: the symbols. The system interacts only with the form of symbols (their physical attributes), not their meaning.

Question # 3: How do I know when such a cognitive system is functioning properly?

Answer: When the symbols appropriately represent some aspect of the real world, and the information processing leads to a successful solution of the problem posed to the system.

The representationist view described above is exemplified by the expert systems approach. Forsyth (1989) describes the four main methods of knowledge representation in expert systems as being rules (in if-then format), semantic nets representing relationships between objects as linguistic links between nodes, frames that act as records to contain data and rules, and Horn clauses which are a form of predicate logic. In all cases there is an inference engine that is separate from the knowledge and which provides a search mechanism to try and find solutions to queries by tracing through the hierarchy of symbolic relationships. Expert systems, regardless of the method of knowledge representation, manipulate symbols to infer information through relationships. They use a hierarchical structure to represent differing levels of knowledge, and above all the individual symbols and relationships are explicit.

An implication of being able to represent intelligence through symbolic manipulation is that the appropriate symbols and their relationships with each other must be identified. In trying to model the expertise of a human in some domain, the symbolism of that person must be captured, a process described as knowledge acquisition. However, Dutta (1993) notes the difficulty in getting people to explain their decision making, and states that “knowledge acquisition is widely recognised today as the bottle-neck, and the most critical part, in the development of knowledge-based systems.

The difficulty of the knowledge acquisition process has been regarded as a symptom of the weakness of the entire cognitivist approach.

Dreyfus *et al.* (1986) take issue with the notion that all decision making can be expressed as clear and rational rules, noting that the capacity of humans to use and

adapt experience can not be captured by a set of flat rules. They lay out five stages of skill acquisition in humans :

1. *Novice*: where facts and features relevant to the skill are recognised, along with some rules for determining actions.
2. *Advanced Beginner*: where performance reaches a marginally acceptable level after the novice has had sufficient experience in dealing with real situations.
3. *Competence*: hierarchical decision making is learned, where the relative importance of features can be assessed, and appropriate goals identified.
4. *Proficiency*: the person starts to recognise situations through previous experience in an intuitive rather than analytic way, but will still think analytically about what to do.
5. *Expertise*: the expert knows what to do from a mature understanding that is developed through experience. The expert uses intuition in recognising a situation and deciding what to do.

Dreyfus *et al.* emphasise that “intuition or know-how ... is neither wild guessing nor supernatural inspiration, but the sort of ability we all use all the time as we go about our everyday tasks”. The implication of this is that the expert has gone beyond the stage that can be represented by rational rules, and that by trying to represent knowledge symbolically, the best that can be achieved is competence.

Pollock (1989) suggests that intuition comes from the use of Q&I (Quick and Inflexible) systems which are responsible for most of the everyday inductive and probabilistic inference which people use. These are subconscious systems that are used to perform fast heuristic calculations. Pollock adds that the heuristics are not "just inflexible but also inaccurate". Pollock goes on to say that true probabilistic reasoning is often computationally infeasible as the number of probabilities increases, so instead people use an approximate probabilistic method. Kahneman *et al.* (1982) describe a number of experiments which back up the claim that very few people use rational probabilistic rules when faced with uncertainty.

In summing up, the key argument about the appropriateness of Classic AI as a representation of intelligence is whether or not there can be a split between the hardware of the brain, and the software of the mind. The cognitivist hypothesis is that the mind can be treated as a separate entity, and its workings represented through the computational manipulation of symbols. The approach is countered by arguments that there is more to intelligence than rational and logical manipulations of symbols, and that intuition through experience plays a heavy part in everyday decision making.

Connectionist AI

Boden (1990) points out that connectionism is a general term which covers a variety of systems, but which have the common features of having simple processing units with parallel connections between them. The connections specify relationships between the individual processing units. Neuroscience and the functioning of the brain and nervous system were the original inspiration for connectionist models, although many models function in a significantly different manner from the brain. The most commonly used connectionist models are neural networks.

McClelland *et al.* (1986) state that the reason why people are smarter than machines is that the brain's computational architecture is much better at natural information processing tasks. Connectionism tries to capture some of the features of the brain's computational architecture. Hinton *et al.* (1986) point out that while the cognitivist symbolic approaches use local representations of relationships between symbols, neural network models use representations that are distributed across the processing units, and that each unit is involved in representing several concepts.

The knowledge captured by a neural network is very much tied up in its structure. Willshaw (1994) notes that a key feature which has created interest in neural networks is the development of algorithms that allow them to effectively learn from examples. Thompson (1993) notes the interest caused by NETalk (Senjowski & Rosenberg, 1987), a program which learned to pronounce english, and that exhibited a number of similarities with children learning to talk.

Varela (1992) points out that connectionist models have shown some basic cognitive capacities. Varela considers the main features connectionist approach using the same questions as those used earlier with cognitivism :

Question # 1: What is cognition:

Answer: The emergence of global states in a network of simple components.

Question # 2: How does it work?

Answer: Through local rules for individual operation, and rules for changes in connectivity between the elements.

Question # 3: How do I know when such a cognitive system is functioning properly?

Answer: When the emergent properties (and resulting structures) can be seen to correspond to a specific cognitive capacity: a successful solution of a required task.

The distributed nature of representation in neural networks means that the explicitness of relationships is lost among the connections and processing units of the network. This means that the effectiveness of the representations can only be judged from the outward performance of the network, and not the internal structure. Due to this feature, Turban (1992) identifies the lack of good explanation facilities as a weakness of neural networks.

Despite some of the features of neural networks that stem from the workings of the brain, it is generally accepted that neural networks are far from getting anywhere close to the power of the brain. Therefore, while the networks are being used for limited tasks, they are not capable brain-like functionality.

Varela (1992) notes that a weakness in the way that neural networks are used (at present) is that they, like cognitivism, are representational. This is because “a criteria for cognition is a successful representation of an external world which is pre-given, usually as a problem solving situation”. Varela points out that true cognition is in

recognising the context in which action is required, which is brought forth by real world interactions rather than being pre-given.

Embodiment

Varela (1992) puts forward the idea of embodiment, where behaviour is bound up in the environment that the entity exists in. It is argued that using symbolic abstractions, or formulating a situation so that it can be represented by a connectionist model, involves separating the behaviour from the context in which it was developed.

An approach that falls within the boundaries of the notion of embodiment is the concept of robots that live in, and react to, the real world, rather than using representational abstractions. For instance, Brooks (1991) rejects the representation approach of classical AI stating that it involves searching through a set of abstractions that the programmer has developed and exists in the abstract rather than the real world. Instead Brooks talks about the development of ‘Creatures’ that are made up competing low-level activities which allow them to react to their environment. There are no hierarchical representations or abstractions, but out of the set of competing behaviours, a coherent pattern of behaviour emerges. This is not a connectionist approach, since the activities are individually engineered and have few connections between them.

Boden (1994) notes that while the situated robots offer a novel approach that may lead to useful forms of artificial life, the approach is unlikely to lead to an explanation or representation of human intelligence.

2.3.5 Implications of AI Paradigms for Intelligent Agents in Simulation

As noted in Section 2.3.2, in the simulation context, the interest is in the ‘soft AI’ approach, that is representing in the model the outward appearance of intelligence to determine how that impacts on the operation of the system of interest. Much of the discussion on paradigms for AI is from the ‘hard’ context of representing the workings of intelligence, however, many of the issues raised still have an impact on the less ambitious aims of the simulation context.

All of the paradigms have strengths and weaknesses in terms of their ability to represent intelligent decision making and in their practical use. In the end there must be trade off between what is desirable, and what is achievable. To this end, it is necessary to determine which features are most important to us for the simulation models, and to identify what the various methods of representation can actually, or at least have the potential to, deliver in relation to those features. The issues discussed will play an important part in the following analysis.

2.4 Desirable Features of Approaches for Representing Decision Making

In order to evaluate approaches for representing intelligent agents and the potential of alternative approaches, it is necessary to determine the features that would be desirable for an approach. This takes into account the issues raised in Section 2.3, i.e. the aims in representing intelligent decision making, the types of decision makers to be modelled, and the arguments for and against the different paradigms for AI.

A number of desirable features for approaches to representing intelligent decision making are discussed below. It is unlikely that any one approach will satisfy all of the desirable features, so it is necessary to consider which ones are most crucial.

2.4.1 Accuracy of Representation

Thesen & Travis (1992) state that simulation involves the representation of a system at the required level of detail. This level of detail is dependent on the required accuracy of the results, the amount of time available for modelling, and accuracy of information used to build the model. As a part of the simulation, the representation of an intelligent agent should similarly be able to capture the effects on the decision of all criteria that can be observed within the simulation, to the desired level of accuracy.

In addition, it is necessary to consider the representation of decisions that are a choice from a discrete set of possible decisions, and those that involve the setting of a continuous control parameter.

2.4.2 Ability to Represent Variety in Decisions

The approach should cater for variation in peoples decision making, as described for Customer agents. Thus, the representation should allow for different decisions to be made for a particular situation if this is seen to occur in the real system. An ability for this sort of representation fits in well with the simulation approach in that it accepts the possibility of variability and allows a (possibly stochastic) mechanism to handle it.

The ability to represent variability in decisions may to some extent remove the need to model in detail peoples reactions to uncertainty (see 2.4.6). As noted by Kahneman *et al.* (1982), people react differently to uncertainty, and this could be picked up through a mechanism to represent variability.

2.4.3 Data Driven Model Development

In the Customer archetype, it was noted that people were not always available to discuss their decision making.

In representing decision making in a simulation, it may often be the case that there is an attempt to model the observed behaviour of people, without being able to go through a process of trying to get them to state the rules for their decisions. An example is the decision making of customers in a shop. Here it may be able to ask them a few questions, but it is unlikely that they will be available long enough for a rigorous process of knowledge elicitation. Even if they are available, it may be difficult to capture the logic of everyday decision making, which is likely to be less cognitive than more unusual and deliberate forms of decision making.

Bearing these problems in mind, it is desirable to be able to represent decision making, at least partly on the basis of observed behaviour. Such a data driven approach would be much in keeping with that used by simulation developers for representing stochastic processes, as described by Law & Kelton (1991).

2.4.4 Clarity of Representation

A commonly desired feature of knowledge-based systems, as described by Forsyth (1989), is that the representation of knowledge should be structured and distinct from the procedural code, so as promote clarity and ease of modification. Clarity is deemed to be an important strength by the classical AI community. An approach to represent the behaviour of intelligent agents in a simulation similarly requires ease of modification for the development and maintenance of the model, and thus clarity should be an aim.

2.4.5 Explanation of Decision Outcome

Williams et al (1989), in developing a simulation model which contained scheduling rules, noted the desirability of having an explanation of the reasons for the decisions that the system was making (although their system had no such feature). A facility to explain the decision making may help in the validation of the simulation model or enhance the degree of learning from observation of the simulation.

2.4.6 Allowance for Uncertainty

Decisions may be made based on partial knowledge of the criteria. For instance where possible future events are taken into account, but where the exact nature of these events is uncertain (the system having stochastic elements). Alternatively there may be cases where perception or measurement error does not allow a consistent or accurate knowledge of the decision criteria. Kahneman *et al.* (1982) demonstrate that these uncertainties can have a significant effect on decision making. Thus a system of representation that makes some allowance for these uncertainties is desirable.

2.5 Current Representations of Intelligent Decision Making

The following presents the main approaches that are currently being used to represent intelligent behaviour in simulation models. The advantages and drawbacks of the approaches are considered in detail in Section 2.7, while an overall comparison is made in Section 2.8.

Ören (1994) gives a review of the use of artificial intelligence in simulation. While there has been considerable work in the area of intelligent simulation environments and advisors, it includes little on the use of intelligent agents in simulation.

The following concentrates particularly on the issue of representing intelligent decision making within the simulated system. Note that most of the papers do not appear in the review by Ören.

2.5.1 Abstraction

Fishwick (1988) describes abstraction as "a method for translating one process description into another more abstract one". This generally involves a simplification in terms of the system components and the complexity of their behaviour. In some senses, all simulation modelling involves abstraction since there is only concentration on those aspects of the system that are of most interest, reducing the scope of model and the behaviours of the entities. O'Keefe and Roach (1987) state : "Human thinking ... can reason about a system over a number of different levels of abstraction. It is also possible to simulate over these various levels". For this discussion, abstraction of intelligent behaviour is taken as representing a significant reduction in the complexity of the decision making model.

The complexity of the decision may be reduced by making assumptions about the behaviour of the intelligent agents. This can involve ignoring or combining factors so that the decision criteria are simplified and can be represented by a few rules. Further simplifications can be made if all the intelligent agents are assumed to act in the same way. For instance, shoppers who all take the same route around a supermarket, or always choose the shortest queue for the checkouts, or as in a model described by Warnshuis (1967), drivers on a road who have caught up with the car in front always follow at exactly the same distance relative to speed until able to overtake.

An alternative solution where different behaviours are observed is to represent the whole decision making process by a probability distribution. Here the effects of all the criteria for making the decision are treated as residual (or random) variation so that the

final decision is based purely on the value obtained from sampling from the distribution. For instance, telephone orders or enquiries at a dispatching office may need particular actions by the member of staff taking the call depending on the nature of the call and circumstances in the office and warehouse. However, the time taken to deal with each call may be determined by simply sampling from a probability distribution.

The key aspect of abstraction is that the decision making is being simplified to an extent that it is capable of being modelled in the same way as any other simulation object. This is exemplified by Stern & Sinuany-Stern (1989) in modelling evacuation from urban areas. They use simple flow charts for decision making using probabilities to determine movement through the chart. There is some consideration of criteria which may effect the decision, although these tend to be global factors such as whether it is daytime or night.

2.5.2 Knowledge-Based Systems

Knowledge-based systems are considered to be the mainstay of the classical AI paradigm. All decision making and behavioural processes are viewed as being through the application of explicit logical rules. Even in cases where there is an acceptance of unreasoned behaviour, some consider knowledge-based systems to be able to represent the outward appearance of at least competent decision making.

2.5.2.1 Defining a Knowledge-Based System

Jackson (1990) defines a knowledge-based system as "any system which performs a task by applying rules of thumb to a symbolic representation of knowledge, instead of employing more algorithmic or statistical methods". Jackson gives a tighter definition of expert system, being a subset of knowledge-based systems and requiring 'expert' knowledge, and being used to "solve problems of genuine scientific or commercial interest". Turban (1992) draws a similar distinction between expert systems and knowledge-based systems, but concedes that the terminology is not widely accepted, stating that "the terms ... are frequently used interchangeably". Shannon *et al* (1985) give the definition: "'Expert' or knowledge-based systems are designed to compile the

experience of any number of experts in a given field into a series of rules". Shannon *et al* regard expert and knowledge-based systems as being synonymous, with the single definition falling between the two given by Jackson.

Forsyth (1989) states that in a knowledge-based system there is a separation between the knowledge base (the facts and rules) and the inference engine (the actual search through and use of rules for reasoning). This representation allows more complicated rules to be written without having to worry about the mechanism for interrogation. This is a common form for knowledge-based systems, and more particularly expert systems but not a necessary one according to the Jackson definition of a knowledge-based system.

In the following discussion, the Jackson definition of a knowledge-based system will be used. These are systems which use formalised logic, usually rules to represent behaviour. As with abstraction, it could be argued that all simulation models are knowledge-based systems in that they contain rules about the logic of interactions in the system (Doukidis, 1987). It is useful to draw a distinction from this to concentrate only on the representation of decision making.

In addition, it is desirable that the representation of the knowledge of decision makers is in some way separated from the rest of the logic in the simulation model in order to provide accessibility and clarity of this knowledge. However, this does not necessarily require the separation of knowledge-base and inference engine as described by Forsyth.

In the discussion, the terms knowledge-based system and expert system will be used interchangeably, depending on the terminology of the referenced work. These terms can be regarded as being synonymous unless specifically stated. The actual contents of the knowledge-base may contain 'expert' knowledge, but this is not a necessary condition or even desirable in cases where there is a wish to represent the actions of ordinary people who may not be 'expert' in making that decision.

2.5.2.2 Combining Knowledge-Based Systems with Simulation

O'Keefe (1986) suggests a number of ways in which an expert system can be combined with a simulation model. These include embedded, parallel and co-operative models which will be used as classifications in the following discussion. The fourth type of combination is an intelligent front end to a simulation, which does not involve intelligent agents and so is outside the scope of this discussion.

Embedding Knowledge-Based Systems within Simulation Models

The knowledge-based system is embedded within the simulation software, and called upon when some decision or behaviour needs to be resolved. This is the most obvious starting point for resolving the behaviour of intelligent agents using a knowledge-based system. The traditional simulation model would control the time clock and scheduler and so would be a natural choice as the controlling system, calling on the rule-base when a decision needs to be resolved and supplying the necessary information for the decision to be made.

Knapp *et al.* (1987) describe the Submarine Interactive Attack Model (SIAM) for the simulation and analysis of engagements between two submarines. The system is designed to analyse the effects of the submarine hardware, processing of data and tactical decision making. The tactics are implemented by defining conditional rules for the action to be taken in a perceived tactical situation. These rules are implemented in an ordinary procedural language as part of the SIAM system. The rules can have combined conditions (AND, OR) and have a priority order. The rules are scanned and the rule which has all conditions satisfied and has the highest priority is chosen. The actual rules themselves are coded as input data for the construction of decision trees. SIAM is not using a conventional expert system with separate knowledge-base and inference engine, but the system undoubtedly contains an embedded knowledge-based system.

Williams *et al.* (1989) describe a simulation model for the supply replenishment of warships at sea. A rule-based advisor was embedded into the system to make decisions about scheduling the transport force. Initial versions of the model required a

user to do much of the decision making. Knowledge acquisition was done by taking commanders, who had some experience with such logistical problems, through a series of scenarios on the simulator and getting them to verbalise their decision making process. The decisions were also recorded by the simulation program and were analysed after the session. The rule-base, called ADMIRAL (Automated Decision-Maker In Replenishment And Logistics) was implemented using FORTRAN, as a sub-component of the simulation system. The rules were designed to be relatively easy to find and change. Rules were given priority levels related to their perceived tactical importance. Communication with the rule-base was via a Daily Order Scheduler which compiled a logistical schedule for the day, this in turn was in communication with the simulation. This system can be thought of as having a separate rule-base (ADMIRAL) and inference engine (daily order scheduler) and uses expert knowledge. It does not however contain an explanatory interface for the decisions that are made.

Both systems with the knowledge-base embedded within the simulation system make attempts to separate knowledge for decision making into blocks which are distinct from the simulation logic. This is certainly a sensible approach for handling sets of rules that are larger and more complex than the standard simulation logic. The separation allows the rules to be found, reviewed and changed more easily.

It is implicit in the approach of adding a rule-base to the simulation system, using the same procedural language, that the rule manipulation will not be as advanced as in most languages and shells specifically designed for creating knowledge-based systems. Since the decision making domain is usually fairly specific, and can be broken down into several different areas allowing concentration on a limited number of rules, this approach can still be effective. It also avoids the problems of linking and incompatibility that can occur when combining several systems using different packages. The ease of creating, verifying and maintaining the rule-base is likely to deteriorate as the number and scope of rules increase. Thus, the approach may not be suitable for particularly complex and wide ranging models of decision making, or for making use of parts of rule-bases which already exist.

Ozel (1992) also has a knowledge-based system incorporated into the simulation, with all the coding done in Fortran. The model is used to simulate the behaviour of humans who are in burning buildings. The simulation is time-slice based, with the model being updated generally every 2-6 seconds, depending on the type of situation being modelled. At each point the decisions, actions and progress of people in the building are determined. People in the building are given physical attributes such as speed of movement, location in the building and tolerance to smoke. There are also psychological attributes such as tolerance to stress and knowledge of building layout. People are divided into occupant groups and are assigned the attributes for that particular group.

The simulation model contains a decision and action generator module. Behaviour in the fire is viewed as being episodal with each episode have a particular goal. Selecting a new goal is dependent on the conditions in the building and the characteristics of the person, which are used to determine goal modifiers. These modifiers are used to determine the probabilities of particular goals, which are then selected through random sampling. The goals and probabilities are determined from analysis of actions in a large number of actual fires. Selecting a particular goal then sparks of a sequence of actions that affect the movement of the person and interaction with the environment.

The approach taken by Ozel is interesting in that it clearly uses rules for determining the goal modifiers, but also involves abstraction through the use of stochastic processes. Also this approach is not trying to model the details of the way that people make decisions, but the outward appearance of actions, based on analysis of data. The model does not use a standard inference engine, but requires a specialised mechanism for the manipulation of goal probabilities and look up of action libraries.

Embedding Simulation within a Knowledge-Based System

A common reason for having a simulation model embedded within an expert system is where the simulation is being called by the expert system to help analyse or resolve questions. Although such a system displays intelligence in being able to examine the simulation in some way, it does not involve the modelling of intelligent agents *within*

the simulation, and so will not be considered further. For reference however, an example of such a system is given by Pinkowski (1990), and a review of systems by Merkuryeva & Merkuryev (1994).

A situation where it could be said that a simulation is embedded within an expert system, is where the expert system has control of the simulation environment. Marsh *et al.* (1990) propose a system for the simulation of battles using a mixture of Prolog and Fortran. Here the Prolog is used to implement the logical parts of the simulation: the event logic, battle tactics and the overall simulation control. Fortran is used as low level code for the more numerical parts of the simulation: representation of environmental data (battle ground, force commitments, etc.) and the manipulation of the environmental data (conflict resolutions, etc.).

The expert system component contains rules on the behaviour of units, as well as having overall control of the running of the simulation. The Fortran is then asked to go and resolve the actual outcomes of events based on the simulation state and the behaviour of entities involved, as well as incorporating any random elements. It is argued that this organisation provides a flexible environment for the creation and control of battle simulations. The Prolog provides clarity and is easier to modify than the original Fortran models. At the same time, the greater efficiency of Fortran for numerical manipulation can be retained, and existing Fortran simulation code can be re-used.

Co-operative Systems

An approach for getting closer co-operation between a discrete-event simulation model and a knowledge-based system is to bring elements from both areas into a single system.

Flitman (1986) gives details of a simulation system that was developed using Prolog and has a rule base for decision making. The problem involves modelling an intelligent Automatic Guided Vehicle (A.G.V.) which has to find a route around a network to deliver items from one position to another, with the possibility of carrying several items

at one time. The simulation was developed through the use of high level clauses, which are given specific instances for modelling actual entities and relationships. The flexibility of Prolog allows the rules representing the behaviour of the A.G.V. to be added into the system with relative ease. The rules were based around several symbolic heuristics for planning routes around the network and the system also provides a facility for listing which rules were used with some explanatory text about the rules.

The approach of using an expert systems language to implement a simulation could be argued as special case of the expert system being embedded in a simulation since the simulation still has the control. However, the simulation is being implemented in an environment where the rule-base and inference engine are separated.

Moving further in the direction of co-operation are the intelligent simulation environments. Here the knowledge-based system is being used to aid in the creation of simulation models or in the analysis of simulation results. An example of such a system is given by Reddy *et al.* (1986). Adelsberger (1986) describes a number systems based around an object-oriented rule-based system, which give entities slots that define their behaviour. These slots give an opportunity to model intelligent decisions, although no actual examples of intelligent decision making are given.

Ruiz-Mier and Talavage (1987) give an example of an object oriented, logic programming environment CAYENE which is being used for the control of A.G.V.s in a flexible manufacturing system. An A.G.V. is represented as an object that contains properties of status, load, current position, an internal map of the system and rules on how to behave in the network. The importance of various routes are represented by a weighting for each pair of connected points. The rules are heuristics on how to make use of the map and weights to travel around the system. The A.G.V. can also adapt to the environment through rules that allow changes to be made to the route weightings.

Zeigler (1990) describes the DEVS system which is a knowledge-based simulation environment. The environment is LISP based and uses the object oriented paradigm.

The DEVS environment allows the creation of intelligent agents through the ability to access the LISP language that underlies the system. This means that the intelligent agents object can contain methods (internal procedures) specifying conditional rules for action which respond to the current state of the object, and its awareness of the system state. Zeigler does not present an actual case of intelligent agents being used in practice, but does present a hypothetical model of a robot controlled laboratory dealing with fluid transfer processes.

Deadman & Gimblett (1994) propose a model using the DEVS environment for modelling the use of forest areas by hikers. The forest is represented by a grid of zones, while the hikers are represented by intelligent agents. The hiker agents reside in a particular location and decide which adjacent location to move to next. When they arrive at a location, the satisfaction of their experience is reduced if they are in close proximity to other hiker agents, and increased if they are not. The prototype model only contained simple rules for deciding which location to move to next. Although this led to problems with the behaviour in the model, the authors report that preliminary runs led to insights about how forest management policies led to movement around, and enjoyment of the forest area.

Anderson & Evans (1994) describe the Gensim system which is designed to model the activities of intelligent agents as part of an ecosystem model. The Gensim system is based on LISP with object oriented extensions. It uses event-based simulation, with time slicing to represent continuous processes. The ecosystem is divided up into grids, with each grid representing some aspect of the environment. The intelligent agents each have knowledge bases that contain rules to govern their actions. There are also rules to determine the extent of their perception of the environment which affects their decision making. As in Zeigler's DEVS system, it is the use of LISP with object extensions as the basis for creating the simulation environment that allows the use of a structured knowledge base which is bound up with the intelligent agent object.

Mayrand *et al.* (1993) describe an object-oriented system using intelligent agents in a simulation of a rolling mill. Here the simulation system and agent implementations

were created using Smalltalk. The intelligent agents are required to make decisions regarding scheduling and machine set-ups to derive efficient schedules. The agents only have limited perceptions, but are allowed to communicate with each other to give instructions or information. The rules used by agents were placed in a rule-base, and are used much as in an expert system. The authors note that their approach offers an elegant way of bringing intelligent decision making into production models, but this is at the expense of running speed since Smalltalk is not an efficient language for running the simulations.

Parallel Systems

In O'Keefe's (1990) definition a parallel system is where "simulations and expert systems that are designed, developed and implemented as separate software, in parallel, may interact".

Flitman (1986), and Flitman & Hurriion (1987) describe a simulation model of a flexible manufacturing system that is served by two A.G.V.s. The simulation model was developed using Fortran. The rules for describing the intelligent behaviour of an A.G.V. were implemented using Prolog. The Fortran simulation and Prolog expert system were run in parallel on two separate PCs and communicated via a modem link. The simulation model stored and manipulated the system state. When a decision for an A.G.V. route is required, the simulation calls the expert system to resolve the decision, providing it with appropriate data. The expert system applies its rules and passes the decision back to the simulation.

Both simulation and expert system were implemented as separate pieces of software, although the simulation was built with a dependency on the expert knowledge provided by the expert system. The simulation must be designed to provide the relevant information to the expert system, in the correct format. Any changes of rules in the expert systems that require different information necessitate a corresponding change in the simulation to provide that information. The approach does allow the simulation and rule-based intelligence to be handled using the most appropriate software. Also,

other than the restrictions placed by the information passing, it allows an independence of the internal representations of the models.

Tambe *et al.* (1995) describe a system for representing pilots in military simulations. The work provides an intelligent agents module called TacAir-Soar that links into an interactive simulation environment for battlefields. The battlefield simulation is based on distributed interactive simulation technology, which allows independent simulators to be linked up via a network to form the whole environment. This means that the individual simulators can be developed in the most appropriate manner. The TacAir-Soar system is a rule-based system describing pilot goals and appropriate actions to achieve them. These goals are hierarchical and describe a route for goal actions, starting from intent, through all the subsequent goals to produce action.

Tambe *et al.* note that in developing the intelligent agents, they are only interested in higher level tasks, and not low level perceptual tasks, therefore the information passed to the system is already in the appropriate abstract form. Speed of operation is a key factor since the simulation is real time, so as much data processing as possible is done in the interface part of the system, to provide the rule-based system with only pertinent information. The rule-system was built using C for speed of operation, and the inference engine was highly geared to the problem context for efficiency of operation. Due to the specialised approach, the system cannot be considered to follow the architecture of an expert system, but comes under the wider definition of a knowledge-based system. The system is reported in the paper to be partly operational, having successfully participated in constrained air-combat simulations against human pilots.

2.5.3 Rule Induction

One of the criticisms of the practical application of the knowledge-based system approach is the difficulty of getting experts to describe their process of decision making. Rule induction (or decision tree induction) is suggested as a way of overcoming the inability of experts to express their knowledge, by deriving rules from example data. By looking at past examples of decision making, it is hoped that patterns can be detected which explain the decisions.

Dietterich & Michalski (1984) define inductive learning as "a search for plausible general descriptions (inductive assertions) that explain the given input data and are useful for predicting new data". In other words, the process involves finding rules which explain the decisions made in the example data, in such a way that they can be general enough to also deal sensibly with new situations. The data is made up of a number of examples of past decision making where the inputs represent the criteria that are thought to be used for making the decision, and the output represents the decision that was made in each case.

The aim of most of the inductive approaches to knowledge elicitation is to produce a decision tree which may be expressed as rules in a knowledge-based system. Thus the actual representation of the intelligence is much as for the knowledge-based systems discussed in Section 2.5.2 .

As a data driven technique, the relevance of the rules is not only dependent on the induction technique used, but also the quality of the data. Hart (1989) points out that the criteria variables need to be meaningful, and that care should be taken to identify contradictions and gaps in the data as areas for further analysis.

One of the best known induction algorithms is ID3 (Inductive Dichotomiser) as devised by Quinlan (1983). The aim of the approach is to form a decision tree where each branch represents a classification of the examples into smaller sets based on the value of a particular variable. The decision tree contains sufficient branches so that each final subset contains only examples with the same decision outcome. The basic version of ID3 could only cope with classification variables as inputs. Mingers (1986) describes an adaptation to the algorithm which makes it capable of handling integer and real variables. The output of the decision tree is in terms of one of a specified number of classes, represented as a single variable taking integer values only.

As an approach for representing intelligent decision making in simulation models, rule induction does not provide a complete solution, but instead feeds into the expert

system approach by allowing the development of rules from data on past decisions. Thus it should be considered as an additional component to the expert system representations.

2.5.4 Fuzzy Sets

Fuzzy sets are a branch of mathematics which have been used by some elements of the classical AI paradigm. It allows the use of explicit logical representations but accepts a variety and uncertainty in the use of linguistic terms.

Smithson (1987) explains that a set is fuzzy if “an element can belong partly to it, rather than having to belong completely or not at all. Fuzzy set theory ... begins with the assignment of membership values to elements which are not restricted to 0 (non-membership) or 1 (full membership), but which lie somewhere in the interval from 0 to 1”

Use of fuzzy sets is often linked with inexact linguistic terminology. The concept of tallness generally has an inexact definition, that is, it contains within it implicit gradations, or that people may interpret tall as meaning a different set of heights. For example, a male who is 5'10 may be considered to be tall with a membership value of 0.6. Smithson notes that there is no agreed definition of this membership value, however Dubois & Prade (1980) suggest that it could represent the degree of possibility that someone of a particular height might be judged to be tall.

The fuzzy set theory has a number of operators used to represent logical relations between fuzzy sets. These include logical operators such as *not*, *and*, *or*, as well a set of operators called hedges that represent the effect of terms such as *very*, *not very*, *quite*, etc.

Zadeh (1973) states that “the pervasiveness of fuzziness in human thought processes suggests that much of the logic behind human reasoning is not the traditional two-valued or even multi-valued logic, but a logic with fuzzy truths, fuzzy connectives, and fuzzy rules of inference.” The use of fuzzy sets allows an addition to the knowledge-

based systems approach by representing uncertainty and linguistic vagueness in a way that is not probabilistic, but which Zadeh claims may be closer to approach which humans use.

Dubois & Prade (1980) give a theoretic overview of the use of fuzzy sets for decision making where the “choice of actions depends not only on utility functions but also on the states of nature or on possible or expected consequences of action”. These expected consequences represent uncertainty and can be represented by fuzziness. This allows an approximate representation and calculation of the effects of different criteria for deciding on a particular action, which may reflect human decision making better than the probabilistic models which Kahneman *et al.* (1982) criticise.

Mündermann (1993) describes the use of fuzzy logic for representing decision making of intelligent agents in a highway (Autobahn) simulation. The simulation involves a multi-lane road with several classes of vehicle, and requires the identification of possible future hazards in relation to an autonomous mobile vehicle (AMV). The process involves measuring the state of all relevant vehicles relative to a particular AMV. Case studies and interviews were used to collect knowledge from car drivers about how they predicted the future actions of other drivers. These rules were combined with the data measurements to produce predictions about the behaviour of other vehicles, and if they may be a hazard. The measurements and responses were represented as fuzzy linguistic categories (e.g. future, soon, very soon, much faster, much slower), and combined to produce predictions using fuzzy logic operators and lookup tables of rules. The work as reported, was still in progress so no results on its success were available. The aim was to test the model using data collected from video pictures from an Autobahn.

The approach that Mündermann describes has close links with the knowledge-based systems approach discussed earlier. It uses rules of thumb for decision making, but represents uncertainty and various measures using fuzzy linguistic sets. The approach requires knowledge acquisition, but the fuzzy linguistic representation may make it easier to describe and capture the decision making process.

2.6 Alternative Representations of Decision Making

Further to current methods of representing intelligent decision making, other approaches are discussed here which may have features that warrant further investigation.

2.6.1 Case-Based Reasoning

Dutta (1993) notes that experts often make recall to past experience in making a decision by identifying and adapting patterns that they have seen before. Therefore it may be useful to look at a system for representing decision making that makes use of past cases.

Kolodner (1993), in a study of case-based reasoning states “In general, the second time we solve a problem or do some task is easier than the first because we remember and repeat the previous solution”. The case-based reasoning approach tries to replicate this recall process by basing decisions on what was done for similar cases in the past.

Kolodner identifies two main processes involved in case-based reasoning. The first is recalling old experiences that match most closely to the current situation, and the second is adaptation of the old experiences so that they can be applied to the new situation.

The first stage involves searching through the database of past problems to see which contain features and patterns that make them relevant exemplars for the current situation. The choice may be through a similarity of explicit features of the problem, or through more abstract similarities, such as ratios between features.

The second stage takes solutions or parts of solutions used in the past cases to form an approximate solution for the current situation. The solution is then refined to make it more appropriate using a set of adaptation rules which have been created by the system developer. The adapted solution is then compared against the database to see if any similar solutions were used in the past, and if they demonstrated any flaws. The

solution is then used for the real world problem and is added to the database of historical solutions, along with any comments on its level of success.

In the context of representing intelligent agents in simulation models, the case-based reasoning approach may allow the use of past cases of decision making to determine the decision to be made in a particular simulated situation. Past cases of decision making may be collected from recording the actual decisions made in the real system, or from a simulation prototype model that requires human intervention for the decision making. The main work for the systems developer is in determining the information that needs to be stored to embody the problem situation and the solution, and to devise appropriate adaptation rules to allow new solutions to be developed.

2.6.2 Neural Networks

Neural networks are a family of models which have their origins in representations of the workings of the brain. These models are the mainstay of the connectionist paradigm. Lippmann (1987) gives a description of the main principles in neural networks, and describes some of the main neural network models in use. Rumelhart & McClelland (1986) ground the use of neural networks very much in the cognitive science area, with particular interest in perception and memory. Despite their background in neuroscience and cognition, neural networks have been picked up for their numerical processing powers. Smith (1993) concentrates on the use of neural networks as statistical models, being used for regression, classification, discriminant analysis and time-series forecasting. Widrow *et al.* (1994) describe a number of neural network applications showing that they are used in a range of engineering, statistical, operational research and computer science projects.

The approach is based on the principle of a number of simple processors which are joined together to form more complex structures. The networks gain their power because the strengths of links between the processors can be changed so that the structure of the network can be adapted. As Rumelhart *et al.* (1994) put it “The problem of learning in neural networks is simply the problem of finding a set of connection strengths which allow the network to carry out the desired computation”.

The process of learning is data driven, in that the networks are trained on examples of numerical behaviour, and adapt to try and represent that behaviour more closely.

The most widely used type of neural network is the Multilayer Perceptron (or back-propagation network) as described by Rumelhart *et al.* (1986). These models have application areas in discriminant analysis and regression, being able to cope with combinations of continuous valued and classification variables.

Wildberger (1990) notes the use of neural networks in discriminant analysis for detecting suitcases containing explosives and that they outperform linear discriminant analysis. Curram and Mingers (1994) compare neural networks, linear discriminant analysis and rule induction over a range of data sets and note that neural networks and linear discriminant analysis have similar performance except in cases where there are definite non-linear relationships, where neural networks have the advantage. Both techniques generally performed better than the rule induction.

Hornik (1991) in a theoretical analysis of the multilayer perceptrons, shows that they are universal approximators for continuous valued functions, but does note that they are unlikely to approximate equally well in all cases. The use of neural networks for regression is demonstrated by Hoptroff (1993) for a variety of business problems. Church & Curram (1996) compare the use of neural networks and linear regression models for econometric modelling.

Several authors have made comparisons between neural networks and expert systems. Hawley *et al.* (1990) particularly concentrate on the ability of neural networks to adapt in response to data, whereas expert systems generally require a lengthy knowledge acquisition phase. They also note that neural networks can cope with noisy data while many expert systems rely on absolutely correct data. On the downside, there is a lack of a clearly identifiable structure in the network, which makes it difficult to trace through the logic.

Zahedi (1991) looks at similarities between AI (particularly expert systems) and neural networks, stating that both approaches “aspire to imitate human intelligence”. Interestingly, Zahedi’s view is that expert systems and neural networks are suitable for mainly qualitative problems, saying that they are best “when the data is qualitative, categorical, and complex; when judgement patterns and circumstances play a substantial role in the outcome”. This appears to run against the data driven, numerical view of neural networks as it is usually given, but Zadehi is trying to draw a distinction with ordinary numerical processing (such as calculating a payroll) where fairly simple, specific rules are being followed. It points to the neural network’s ability to determine structure and patterns from data that it is exposed to.

Padgett & Roppel (1992) suggest that neural networks may be of use to simulation analysts and introduce the approach from a simulation perspective. Weinroth & Madey (1990) suggest a number of ways in which neural networks can be combined with simulations. The suggestions involve two which most closely couple the approaches. One is the use of neural networks to capture the behaviour of complex and time-consuming simulation sub-process so as to produce results more quickly. The simulation can be used to provide results for the sub-process from multiple runs which are then used to train the neural network. The network then replaces the sub-process to re-produce results for runs of the larger simulation model. The other approach is to include human expertise in the simulation model by training the network on examples of decision making. In a small prototype system, Weinroth & Madey use data on controlling a machining process using information from four measures to adjust two machine settings, so as to optimise production rates. Little detail is given about the form of the model or the data (except that it is limited), but the authors state that the results are reasonably good.

Hurriion (1993) demonstrates the use of neural networks for representing and learning probability distributions. This involves fitting an inverse cumulative distribution function, by training the neural network using random numbers as the input, and a response variable (time) as an output. The probabilities are derived empirically, being determined by the positions of the data values in the sorted data set. The approach is

demonstrated using a number of empirical and theoretical datasets. Hurion shows that neural networks can be used in a way which fits in well with the simulation approach. While the distributions are simple (having fixed parameters), they do demonstrate a data driven approach to modelling (generally physical) behaviour that takes account of random variation.

2.7 Evaluating The Approaches

The following evaluates the approaches to representing intelligent agents in relation to the desirable features discussed in Section 2.4. A summary of findings, and comparison between the approaches is given in Section 2.8.

2.7.1 Abstraction

Abstraction generally involves a reduction in the complexity of the decision making process. Abstraction can be used if the simplification does not have a profound effect on the performance of the simulation. This means that the part of the system which has been simplified is not of direct interest, and that the loss of accuracy caused by the simplification does not have a significant effect on the overall accuracy of the simulation. The determination of what is significant depends on the purpose of the simulation model.

Depending on the form of the abstraction, it should be possible to represent both discrete and continuous decision outcomes. Simple rules, and sampling from discrete distributions allows the choice of discrete outcomes, while mathematical relationships and sampling from continuous distributions allows continuous outcomes.

Due to the simplicity of the abstraction approach, the representation should have reasonable clarity, although it is usually closely tied in with the rest of the simulation code. The simplicity of the model is unlikely to make an explanation facility necessary. The use of a probability decision to determine the decisions does not provide sufficient information to allow explanation.

The use of simple rules for abstraction does not allow for a data driven model, although a probabilistic representation should be based on the observed frequencies of different decisions. The abstraction is unlikely to go into sufficient detail for an allowance for uncertainty to be made, but a variety in decision making is possible if the decision is sampled from a probability distribution.

Overall, the approach is unlikely to be useful in cases where any more than a simple representation of decision making is required. It may be acceptable for a supermarket simulation in which the exact choice of checkout queue does not have a great effect on the results of the simulation, but is unlikely to be acceptable for a motorway simulation which is looking at events caused by individual vehicles. It can allow for differences in decision making if stochastic sampling is used, although the probabilities of different decision outcomes are unlikely to be particularly affected by the decision criteria.

2.7.2 Knowledge-Based Systems

The knowledge-based system approach uses rules to determine how the criteria which can be observed in the system effect the decision outcome. The management of these rules should allow more complex decision making structures to be built than the abstraction approach.

The rule-based approach is symbolic, and so can deal with discrete decision outcomes but the mechanism is poorer at representing continuous outcomes. Representation of continuous outcomes requires splitting the continuous range into a number of discrete ranges, or for the decision outcome to utilise a mathematical relationship to determine the continuous outcome. For instance, representing the decision for change in acceleration of a car may require the use of classes such as : 'small increase', 'severe braking', etc. or a decision to start braking smoothly which may call a mathematical routine to determine the degree of breaking required to reach a certain speed by a certain distance.

Some knowledge-based system have a representation that allows reasoning under uncertainty. Stefik *et al.* (1983) discuss the method of dealing with uncertainty in one

of the early expert systems, MYCIN. They note that MYCIN used a fairly ad hoc method of combining certainty values together, although they defend this to some extent by pointing out that more formal Bayesian probabilities require the storage of a large amount of data, and becomes unwieldy. Kahneman *et al.* (1982) reject the view that humans follows rules of probability in dealing with uncertainty, reporting experiments which suggest that a more heuristic and subjective approach is used instead. Thus while some mechanisms for dealing with uncertainty exist for the knowledge-based systems, there are questions about whether they actually represent the approach used by humans.

There is a general emphasis in the design of knowledge-based systems on the separation of knowledge from the more procedural aspects of the model to enhance clarity and modifiability. Barrett & Beerel (1988) state that the structured representation of knowledge by rules should allow the relationships to be easily understood, although they do note that large knowledge-bases containing many low level relationships can become difficult to follow. Dutta (1993) claims that frame or object based approaches which bind the rules describing the behaviour of the entity concerned are more comprehensible than the pure rule-based approach.

Forsyth (1989) suggests that rule-based systems should be able to offer an explanation of decisions by indicating those rules which were fired. Such an approach was used by Flitman (1986) in combining knowledge-based systems with simulation. This requires either a specific mechanism to identify the specific rules, or for such a mechanism to be offered as part of the inference process.

The knowledge-based system usually has an emphasis on the development of rules as a manual process of knowledge elicitation from written sources and discussion with 'experts'. Thus the approach is not usually data driven.

The knowledge-based system approach is particularly useful if a knowledge-base already exists for the decision making process, since the work of creating it has already been done. Where the knowledge-base does not already exist, the task of creating it

can be relatively painless if a set of clearly defined rules for making the decision are readily available. This implies that the people involved with the system have clear set of criteria for making that decision which can be expressed in the language of logic.

Unfortunately it is often the case that people do not use clear cognitive reasoning to make their decisions. Haugeland (1981) states that being articulate about a skill is rarely necessary for having that skill. Dutta (1993) notes that experts build up an intuitive approach to problem solving which is formed from experience, and from storing away common patterns that have been observed in the past.

These problems create difficulties in using the expert system approach as it relies on being able to capture the explicit rules in decision making. Buchanan et al (1983) identify knowledge acquisition as the "bottleneck in the construction of expert system". Ten years later Dutta (1993) reflects this view, stating that the task of knowledge acquisition "is widely recognised today as the bottle-neck, and the most critical part in the development of KBS [knowledge-based systems]".

Traditionally knowledge-based systems are used to represent best practice, and thus are not concerned with representing variety in decision making. Variety in decision making, from a knowledge-based systems point of view, implies that people are either perceiving the decision criteria differently, and/or are using different decision rules. A representation of uncertainty may allow a representation of differences in perception, but the use of different decision models is likely to greatly increase the complexity of the knowledge-base and acquisition process.

Ozel (1992) does allow variation in decisions by using a rule-base to determine modifiers which are used in developing probabilities for particular goals and actions. This is actually a mixture between the rule-based and abstraction approaches. Random sampling is being used to determine the final outcome. It should be noted that Ozel uses a specialized rather than a typical knowledge-based system architecture for this model.

2.7.3 Rule Induction

Rule induction can not be regarded as a method of knowledge representation on its own, but is best regarded as an additional component to the expert system approach. Rule induction produces decision trees which can be converted into expert system rules. The evaluation will focus on what rule induction offers additionally to the expert system approach.

In terms of accuracy, rule induction is data driven, and so this is partly down to the choice of variables and the quality of the data. However, as in expert systems, there is potential to capture complex decision making structures.

The induction approach to representing decisions does have some drawbacks. The output from the model is in terms of a single decision classification variable. Multiple outputs, or continuous valued outputs cannot be used, which limits the applicability of the approach to representing some types of decision making.

A major problem with the basic ID3 algorithm is that it only works well when the data contains no noise, i.e. no contradictions or residual variation. Noise in the data tends to lead to large trees with many branches. The result is that the algorithm fits to individual examples rather than the underlying relationships, and so leads to a reduction in the ability of the induction tree to classify new situations. An approach to reducing this problem is to prune the tree. This involves removing branches to simplify the tree and hopefully increase its generality. The cost of the approach is that a certain error rate in classifying the examples used in developing the tree must be accepted, so as to reduce the error rate of classifying new examples. Mingers (1989) describes a number of pruning algorithms, and tests their effectiveness on a variety of data sets.

Trippi & Turban (1990) compare rule induction, discriminant analysis and neural networks on theoretical grounds, showing that they have various strengths and weaknesses. In particular it is noted that rule induction is limited in what it can represent, and can suffer from low accuracy.

Curram & Mingers (1994) compare the use of ID3 (using pruning) against linear discriminant analysis and neural networks using a number of different data sets. The results showed that ID3 was generally less accurate in generalising to new examples than the other two methods.

Rule induction needs definite information in the development of rules and so does not deal well with partial information. In cases where measurement error exists, a classification scale can be used to represent the lack of exact figures.

The results of the rule induction can be converted into rules, and so can be used in a knowledge-base. This adds to the clarity of the knowledge representation. However, Curram and Mingers (1994) show that situations where there are a lot of interactions between and the variables, or where relationships are non-linear, can lead to very 'leafy' trees with a lot of branches. In this case, the rule version of the tree becomes rather messy and difficult to follow.

Since the results will be rules in an knowledge-based system, there is the potential to include an explanation facility showing which rules have been fired. However, as Hart (1989) points out, the rules generated by the induction process are likely to be more difficult to follow than rules developed by more conventional knowledge acquisition approaches.

As a data driven approach, rule induction is able to develop rules from examples. Expertise is still required to decide which variables are used and the format they should take.

The ability to represent variability in decision making is limited. Rule induction does not cope well with noisy data and this would include situations where people are seen to make different decisions under what are apparently the same situations. Pruning methods do offer potential for dealing with variability. As described by Mingers (1989), pruning produces a simplified tree by allowing some examples to be mis-classified. An adaptation of the pruning approach may allow the mis-classifications to

be used as probabilities of each decision, which can then be used as an empirical distribution in the simulation to determine the decision outcome. However, the nature of the way that the tree is built up may make generalisations of the probabilities difficult. This is because the rules contain discrete steps in determining a set of probabilities, but the true probability distribution is likely to be continuous. So a small change in one of the decision criteria may produce a change in the probability distribution for the decision, but this small change might not be captured by one of the discrete rule steps. Therefore some mechanism would be required to interpolate from the rules that do exist. At present, such a mechanism is not obvious.

2.7.4 Fuzzy Sets

The fuzzy set approach allows the use of fuzzy logic rules to determine a decision based on a set of criteria. The fuzzy membership and logic rules allow for a degree of inexactitude in the measurement and consideration of the decision criteria. The fuzziness will result in less accurate results compared to an exact set of rules, but may allow a greater complexity of situation to be modelled, and may according to Zadeh (1973), represent more accurately the approach which humans might use. Fuzzy membership can represent uncertainty about exact measurements, and so can be used to represent cases where uncertainty exists.

The knowledge-based systems approach, combined with fuzzy measurements and operators, should allow a separation of the facts and rules, from the inference engine. The use of functions to represent fuzzy set membership may result in a loss of clarity compared to a conventional knowledge-based system due to the increase in the complexity of the logic relationships. However, where linguistic fuzziness is used the situation might be slightly improved since the interpretation of the rules should be more natural.

The complexity of creating an explanation facility is increased over an ordinary knowledge-based system. Turban (1992) notes that systems using fuzzy sets require far more rules to be fired, each with results “ranging [in linguistic terms] from ‘not-at-all’ to ‘completely’”. An explanation facility would thus have to be able to make sense

of a large number of rules and represent the results sensibly. Simply reciting the set of rules and conditions that were used is likely to produce a mass of output, which would be difficult to interpret. A filtering process on the rules to show only those that had a high impact on the final decision may help to increase the useful information produced by an explanation facility.

The approach requires the development of logical rules, even though the operators might be fuzzy. Thus the relationships need to be modelled in a similar way to knowledge-based systems. The use of fuzzy linguistics may allow a description of decision making under uncertainty, or inexactitude of terminology and meaning to be represented more easily. Dubois & Prade (1980) hold that no generalisable approach exists for determining membership functions, but note that approaches vary from subjective estimations to statistical sample analysis.

Smithson (1987) describes the debate over the similarity between fuzziness and probability. The suggestion that fuzzy sets are merely another form of probabilistic reasoning is refuted. If this is so, then a direct link with the sort of sampling used in simulation to represent random variation is difficult. However, there are sufficient similarities with probability which means that it may still be possible to develop a mechanism that allows the set membership function to be used to represent variability in people's decisions.

2.7.5 Case-Based Reasoning

The case-based reasoning approach uses examples of past decision making, with rules on how to adapt these to new situations. This is partly data driven and partly a rule-based approach.

The overall accuracy of the case-based system is dependent on the library of past cases and the quality of the adaptation rules. For a specific situation, the existence of a very similar past case provides a good exemplar to follow. For a more unique situation, the accuracy is more dependent on how well the adaptation rules interpolate between the

cases that are available. Thus the system is dependent on both a good range of examples, and well produced rules.

One feature of case-based reasoning is that the system develops its database of cases as more problems are presented and feedback given on the solutions. In the simulation application, the system would be required to make repeated decisions, with little chance of feedback, and so once in operation it will not continue to develop its expertise.

The design of the case-based system involves clarity in that the cases are stored in a separate database, while adaptation rules can be stored in a separate rule-base, as with a knowledge-based system. This allows a separation of the data and rules from the mechanisms that utilise them.

A case-based system can have the facility for justifying why certain cases were chosen as exemplars, and why particular adaptation rules were chosen. The explanation is often in the form of the raw rules which were applied for the case, as illustrated in Kolodner (1993, p175-176).

The case-based reasoning approach is data driven in as much as the past cases are a form of data which are an integral part of the solution. The effectiveness of the case-based reasoning is partly dependent on the quantity and quality of the cases. On the other hand, development work is required for the rules to determine which past cases to choose as exemplars, and rules to adapt the solution to the current situation. Some adaptation rules may involve simple interpolation or substitution of values, while other will be very domain specific. Therefore development of the rules requires some understanding of the features and relationships which exist in the cases, and how they can be manipulated to provide solutions. Thus, although the case-based approach is partly data driven, it also requires much domain knowledge about how values can be compared and manipulated.

The case-based system can be used if certain decision factors are unknown, since typical values can be estimated from the past examples. The methods for producing these estimates have to be determined by the developer and may not represent the process by which the decision maker makes estimates.

The biggest weakness in the approach is in dealing with variety in decision making. For case-based reasoning to work effectively, a set of rational past decisions are required. A variety in the solutions effectively represents noise which must be dealt with in some way. Whatever approach is used, the result is likely to be a consistent suggested course of action. This makes it difficult to handle the situation where you wish to allow several possible decisions in order to represent the variety in the decisions which are being made. It may be possible to use adaptation rules that contain stochastic elements which interact with the exemplar cases to produce a variety of results. This would add an extra level of complexity to determining the rules, and may be highly dependent on the case data that is available.

2.7.6 Neural Networks

Neural networks are able to represent numerical relationships between variables. The variables can be real valued or binary classifications. In most simulations, decision criteria which can be captured by the model are likely to fall into these numerical categories. The neural network can have several output variables, each of which can also take on real or classification values. Hornik (1991) shows that theoretically, neural networks can represent any non-linear continuous function, while Curram & Mingers (1994) demonstrate that neural networks can provide good results on classification problems, having particular strengths where there is a high degree of non-linearity in the data. As with any data driven technique, neural networks are still dependent on the quality and quantity of the data, and rely on appropriately chosen explanatory variables. Therefore neural networks should offer the potential to model complex decision making based on several criteria, provided suitable data is available.

Neural networks do not handle uncertainty well in terms of inexact measures or estimates of decision criteria. The network uses exact values for its input variables.

The use of classification variables does allow the use of more categorical representations of values if measurement is inexact.

The model of the decision making is formed by the adapted neural network itself. This is essentially a set of node values and connection strengths which are not easy to interpret. Several authors (Wildberger, 1990; Smith, 1993; Swingler, 1996) suggest ways of generating a greater understanding of the behaviour being represented by the network, although all of these are black box approaches which look at the effects that changes to the input variables have on the output variables. On the other hand, the neural network is a self contained module and so it can be kept distinct from the procedural code of the simulation.

As indicated by the discussion on clarity, the neural network would not have a strong mechanism for explaining the decision outcome. It may be possible to measure the relative importance of the decision criteria in coming to the decision, but not to go into the detailed structure of the model.

It should be clear from the explanation of neural networks, that they are a data driven approach. Thus the model of decision making is built up from examples of behaviour. This requires that there is sufficient data available, and that it covers a wide range of circumstances. In a comparison with conventional statistical tools, Wildberger (1990) points out that neural networks are better at interpolation than extrapolation. It is likely that rare but extreme circumstances will not be represented in the training data, so care must be taken to handle these, either during network training or in using the results of the network.

Neural networks offer good potential for representing variety in decision making. Hurion (1993) shows that neural networks can be used to represent inverse cumulative probability distributions, using a random number as the input to the model, to produce the output. In cases of mutually exclusive decision categories, it is possible to represent these as probabilities of occurrence rather than straight binary classifications.

2.8 Comparison of Approaches

Section 2.8 evaluated each of the approaches in terms of desirable features for a method of representing intelligent decision making. This section will compare the approaches in terms of strengths and weaknesses, and identify which approach has the best potential for representing intelligent agents in simulations models.

2.8.1 Summary of Strengths and Weaknesses of Approaches

Table 2.1 shows a summary of the performance or potential of each of the approaches for representing intelligent decision making, with regard to the desirable features set out in Section 2.4. The grading of methods is based on the previous evaluation of the approaches, but still contains an element of subjectivity. Therefore the grading scale is intentionally a coarse one.

The summary shows that each of the approaches has different strengths and weaknesses, and that no one approach is dominant across all the criteria. Given that, it is necessary to determine which of the features are most necessary or important for representing intelligent decision making in simulations, and which can be considered to be less important.

	Abstract.	Expert System	Rule Induction	Fuzzy Sets	Case-based Reasoning	Neural Networks
Accuracy	-	+	+	+	+	+
Variety	+	-	-	+	-	++
Data Driven	-		++	-	-	++
Clarity	-	+	-	+	+	-
Explanation	-	+	-	-	+	-
Uncertainty	-	+	-	++	+	-

Key : ++ Strong, + Reasonable,
- Weak, <blank> Unable

Table 2.1 : Strengths and Weaknesses of Approaches to Representing Intelligent Decision Making

The importance of the features is dependent on the problem context, i.e. the required accuracy of the model, the information already available to the modeller (data, rules etc.), the expertise of the modeller, and the nature of the decision making which is required. For instance, in some cases it will not be considered important enough to get an accurate representation of intelligent decision making to go to any great effort in the modelling. In other cases, a model of decision making might already exist and so that would be a major consideration in the approach to use.

In consideration of the criteria it is assumed that some effort to represent intelligent decision making is thought appropriate, and there are no easy, low cost solutions already available.

One key ability that is of interest in this thesis is that the approach should be able to model the decision making with a good degree of accuracy. What is considered good is relative to the requirements of the simulation model, but here it is relative to what is achievable with the current state of theory and technology. This excludes abstraction from further consideration, although in cases where accuracy is not important, it is likely to be the simplest and quickest to use, particularly as it should be familiar to most simulation specialists.

The concept of variability in the behaviour of entities, whether they be machinery, servers, customers etc. is central to most simulation modelling. The possibility of representing variability in decision making fits well into this way of thinking. This is particularly the case for customer type entities. It is a feature of the abstraction approach which is commonly used in simulations. Fuzzy sets may be adapted slightly to offer variability, but it is neural networks which have the best potential to display this sort of feature, through the flexibility in the relationships that they can represent.

Clarity of representation is useful in model development and maintenance. This is an advantage of expert systems, fuzzy sets and to some extent case-based reasoning. However, each of them requires an inference engine to utilise the knowledge. Rule induction does not provide easy to read rules, although as in expert systems, the

knowledge-base is separate from the procedural code. Neural networks do not provide easy to interpret knowledge. However, the network can be treated as a self-contained object that can be kept separate from the procedural code of the simulation. Therefore, the ease of use of the approaches in the simulation model is dependent on the software available, but the more rule-based approaches are generally easier to interpret.

The availability of an explanation facility is useful during the development phase for verification and validation of the simulation model, but is likely to interfere with the running speed of the finished model. Knowledge-based systems and, to a lesser extent, case-based reasoning can have an explanation facility that would help with white box validation of the model (that is examining the detail of the workings of the model), although the other methods should still allow black box validation.

The usefulness of the data driven approaches depends on the nature of the information available. If rules are available or there is a wish to develop a set of explicit rules, then the rule-based approaches are more applicable. If data of decisions is available, or easier to collect than using conventional knowledge acquisition methods, then the rule induction and neural network approaches are applicable. In the case of customer behaviour, it is likely to be difficult to use conventional knowledge acquisition since customers would not be willing or able to explain their behaviour, and are unlikely to offer the time to use more searching knowledge acquisition methods.

An allowance for uncertainty is a useful feature and is where fuzzy sets have particular advantage. Certain expert system approaches allow some uncertainty with the use of certainty factors or linguistic models. The lack of uncertainty need not be a particular disadvantage if the approach can represent variety in decision making, since different responses to uncertainty can be picked up by that variation. This is particularly the case for neural networks.

2.8.2 Potential for Further Investigation

The consideration of abstraction shows that it has limited potential for representing decision making with a reasonable degree of accuracy. The other approaches show some potential in that they are specifically designed to provide at least the outward appearance of intelligence. The analysis has shown that the ability to represent classification and continuous control decisions differs in the methods. The flexibility of the neural network approach is an advantage over the more symbolic approaches, although none of the others are ruled out of the consideration on this basis. The ability to represent variety in decision making is an important feature which fits in well with the simulation philosophy. In this respect, neural networks appear to have the best potential. This feature also partly makes up for the poor ability to model uncertainty about the values of the decision criteria. Neural networks are also a data driven approach, which may be useful in those situations where decision making can be observed, but detailed investigation of the cognitive processes is difficult. The poor explanation facilities are a disadvantage of neural networks, although some black box validation should be possible.

Fuzzy logic also has some potential for representing variability and has the advantage of greater clarity for interpretation of the model. However, model development could be difficult, particularly given the arguments of Kahneman *et al.* (1982) about the inconsistency and lack of probabilistic rules in human reasoning.

The other modelling methods are rejected because of the difficulties in representing variety in decision making, and the difficulty in capturing the knowledge. This does not mean that they are not useful in cases where consistent, expert decision making needs to be modelled and where the effort in developing the rules is thought to be worthwhile.

On balance, the initial analysis of the approaches suggests that neural networks show good potential for demonstrating the features of interest, and will be investigated in more detail in subsequent chapters.

Chapter 3

The Practical Use of Neural Networks with a View to the Representation of Intelligent Decision Making

The analysis in Chapter 2 identified a neural network approach involving stochastic elements as an area with potential for representing intelligent decision making. The aims of this chapter are three-fold.

Firstly, the thesis is written with the perspective of a simulation analyst in mind, and so does not assume a deep knowledge of neural networks. Therefore one aim is to provide the reader with a grounding in neural networks.

Secondly, no strong methodology has been developed for neural networks. The use of neural networks is still treated as somewhat of a 'black art'. The approaches suggested in the literature for using neural networks differ depending on the background of the researcher and the problem for which it is being used. So another aim is to try to draw a number of the strands of advice together to form a set of general guidelines to be used, or at least considered, for the application of neural networks in this thesis. This is to some extent helped by the publication of a book by Swingler (1996) which aims to pull together lessons learned by a number of neural network users.

Finally, a strong aspect of the use of neural networks in this thesis is the representation of stochastic processes in decision making. The last part of the chapter considers a number of ways in which this representation might be achieved.

Section 1 gives a brief history of the development of neural networks and some of the principles in their construction. Section 2 concentrates more on the features of one type of neural network, the multilayer perceptron, looking at the algebra behind its construction. This is followed by a look at the sorts of statistical applications for

which the network is used. Section 3 looks at the practical issues involved in using the multilayer perceptron. Finally Section 4 discusses possible uses of the multilayer perceptron for representing intelligent decision making where variability in decisions is allowed.

3.1 The Principles of Neural Networks

3.1.1 The Origins of Artificial Neural Networks

Neural networks are a family of numerical processors which are roughly based round the ideas of the workings of the brain. The common feature is that they use simple processing units which are connected together to form representative mathematical structures. The real power of neural networks is the ability to change the strength of connections between processors in response to example data so as to adapt the structure of the network to reflect the patterns inherent in that data.

Thompson (1993) gives a description of the human brain. The human brain is made up of around 100 billion (10^{11}) processors or neurons. Each neuron can have several thousand synaptic connections to other neurons, giving a huge number of possible combinations of synaptic connections in the brain. The major synaptic connections are developed before birth, but it is thought that many connections are formed and reformed throughout life. Communication between neurons is via electrical signals along the neuron cell bodies, and chemical signals at synaptic connections. Although the power of an individual neuron is limited, the massive connectivism in the brain gives it immense potential.

Compared to the dimensions and power of the human brain, artificial neural networks are severely limited. Rather than emulating the brain, the development of neural networks has been along the lines of aiming for more modest mathematical models.

Hagan *et al.* (1995) give a history of the development of neural networks. Many of the early papers can be found in a collection edited by Anderson & Rosenfeld (1988).

The first view of artificial neural networks started with McCulloch & Pitts (1943). In this paper they developed a logical model for the operation of a neuron. This neuron has two states : active or inactive and has several synaptic connections. Inputs are received through the synapses and are summed, with the neuron being activated if the sum reaches a certain threshold. Implicit in the paper is the notion that a single neuron is a simple processing device, but the power comes from the connection of a number of these neurons.

Rosenblatt (1958) developed the first practical neural network, the *perceptron*. This was a network of neurons organised into layers, with a learning algorithm for updating the connection strengths between neurons. This network was developed to perform pattern recognition. Shortly afterwards Widrow & Hoff (1960) developed a network with similar properties to the perceptron, but with a rather more elegant learning rule. The principles of the Widrow-Hoff learning algorithm have been carried forward to many of the most popular modern neural network approaches.

Much of the enthusiasm for neural networks died down with the realisation that the results produced by neural networks were not as good as had been hoped. Learning of small problems using small networks was reasonable, but larger networks were much less successful. The decline of neural networks at the time was precipitated by the publication of *Perceptrons* by Minsky & Papert (1969). Here they demonstrated mathematically that the Rosenblatt and Widrow-Hoff networks suffered from considerable limitations in the sorts of problems they could tackle.

Some work with neural networks still continued, with Kohonen (1972) and Anderson (1972) producing models for associative memory. These are models which are capable of associating patterns that frequently occur together.

A resurgence of interest in neural networks grew with two key pieces of work. Hopfield (1982) developed a new theory and explanation of the operation of associative networks. Rumelhart *et al.* (1986) developed a new backpropagation algorithm for training perceptrons with multiple layers. This algorithm overcame many

of the criticisms made by Minsky & Papert (1969), and has formed the basis for the most popularly used form of network today, the multilayer perceptron.

3.1.2 The Structure of a Neural Network

The basic building block of an artificial neural network is a neuron. Figure 3.1 shows a biological neuron. The neuron collects input signals through the dendrites, which form the interfaces with other neurons for passing on signals. The combined signals from other neurons are passed onto the soma of the neuron, which acts as the processor. If the strength of the signals reach some threshold then the neuron is activated and passes a signal onto other neurons down the axonal path. The sensitivity of the neuron is controlled by its activation threshold.

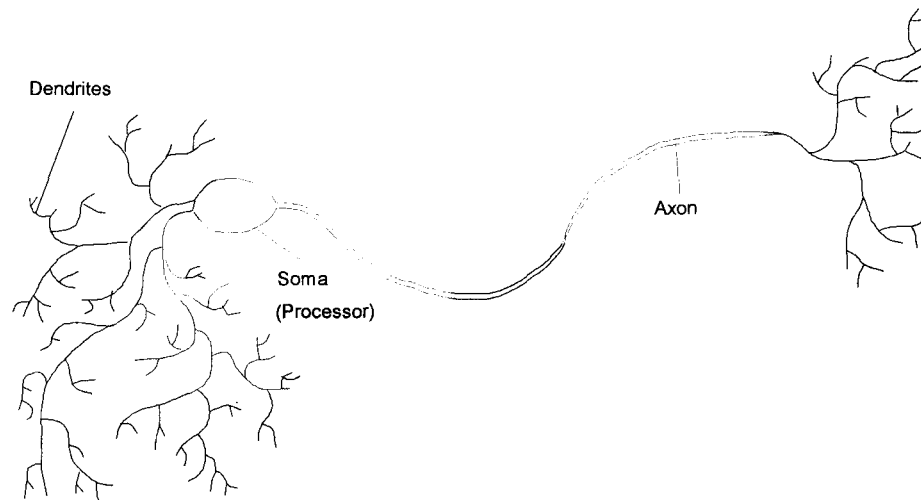


Figure 3.1 : A Biological Neuron

Figure 3.2 shows a typical artificial neuron. It, like its biological counterpart, is made up of three main parts : input connections, a processor, and output connections.

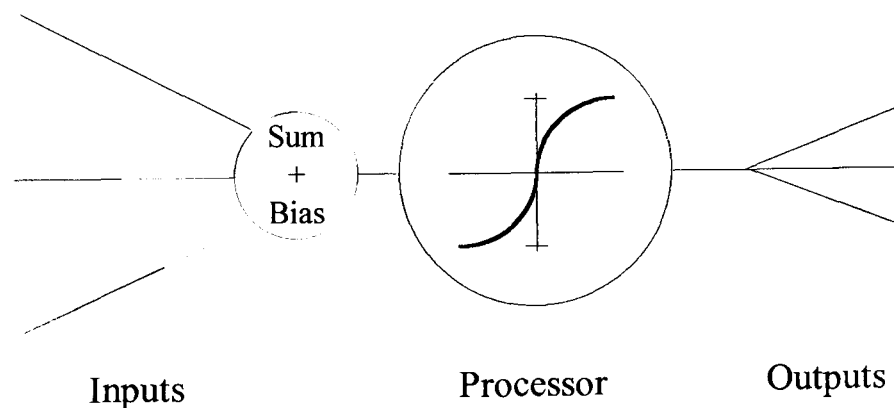


Figure 3.2 : A Typical Artificial Neuron

The inputs are weighted connections from other neurons. These weights represent the strength of connections between neurons, and can generally take positive values (excitatory) and negative values (inhibitory). The inputs are summed together, and added to a bias term to form a net input value. The bias represents the overall sensitivity of the neuron. A large negative bias means that the neuron is difficult to activate and requires large input values to do so, while a large positive bias means that the neuron is easy to activate, and may need inhibitory inputs to prevent it being activated.

The net input value is then passed onto the activation function. This function can take on a variety of forms depending on the type of neural network, the most common being the sigmoid (s-shape), hard-limiting and threshold functions, as shown in Figure 3.3. In most forms of network, the activation function will produce an output in a specific range, 0 to 1 or -1 to 1 being the most common.

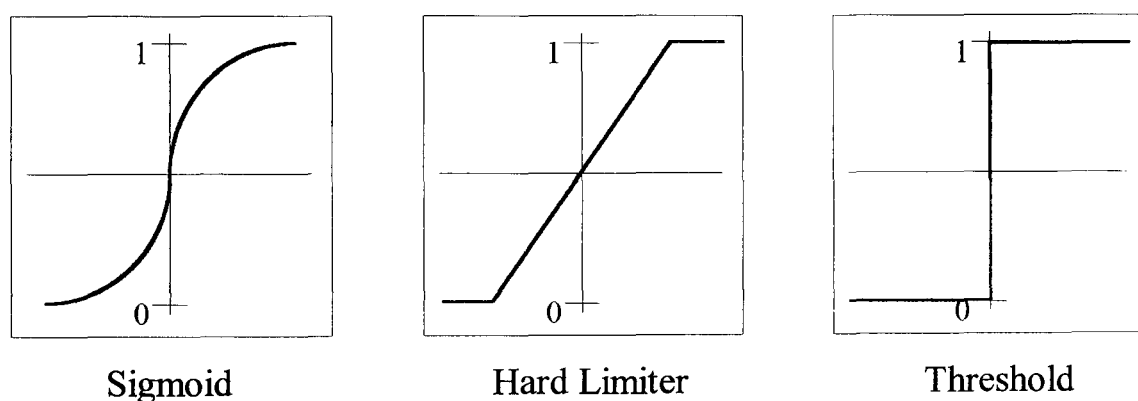


Figure 3.3 : Typical Neuron Activation Functions

The activation value of the neuron is passed on through the network by the output connections. The output connections may feed into other neurons, or they may pass out of the neural network as final outputs (i.e. responses) of the network.

Some networks can be highly interconnected, where potentially every neuron has connections to every other neuron. An example of a highly connected model is *The Jets and The Sharks* model shown by McClelland *et al.* (1986, p28). However, most

neural networks are organised into layers where each neuron in one layer has connections to every neuron in the next layer only. Figure 3.4 shows an example of a layered network.

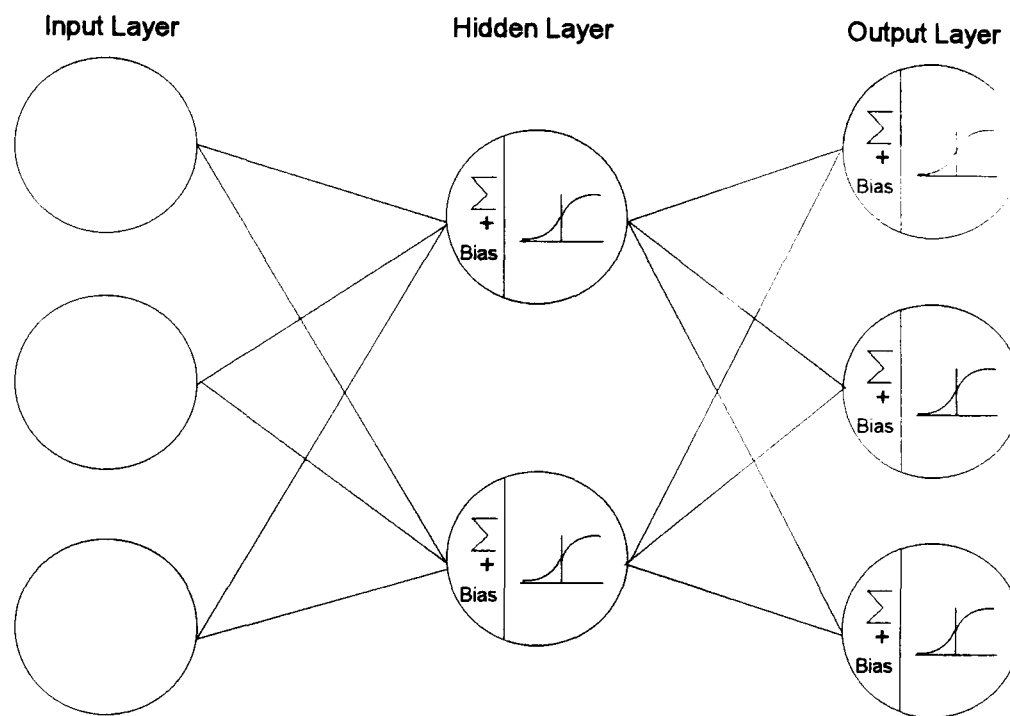


Figure 3.4 : A Three Layered Neural Network Architecture

The input layer of the network takes in the explanatory data (in statistical terms), that is, the data whose behaviour the neural network will react to. The output layer is the network's response to the data, through which the results are passed out of the network. Invariably there are one or more layers in between the inputs and outputs, known as the hidden layers. These do not communicate with the outside world, but are used as internal processors for capturing more complex patterns in the data. Lippmann (1987) demonstrates the effects of different numbers of hidden layers on the ability of the multilayer perceptron to do classification tasks.

The behaviour and patterns in the data are represented by the strengths of connections (weights) between the neurons, and in many cases the sensitivity (bias) of individual neurons. In order for the network to learn, a training algorithm is required to change these weights so as to represent the behaviour better.

3.1.3 Neural Network Training Paradigms

Many different learning algorithms exist for adapting the weights of neural networks to better represent the behaviour of the data. A number of these algorithms are shown by Hagan *et al.* (1995). However, all of the algorithms fall into one of two training paradigms, either *supervised* or *unsupervised* training, or in a few cases a hybrid of the two approaches.

Supervised training involves showing input data to the network, along with a target solution. The response of the neural network to the input data is compared with the targets outputs, and a cost function is used to determine how close the network is to capturing the behaviour of the data. The network connection strengths are then adapted so that the cost function should be reduced, and the network gets closer to the target solution. This approach has similarities with regression analysis, where you are trying to use a set of explanatory variables (inputs) to model to the effects on the dependent variables (outputs).

Unsupervised training involves presenting the network with input patterns but no corresponding target output values. Instead the aim is to associate input data examples which have similar properties, and learn to give them similar output values. In most networks this is similar to performing a clustering operation where you are trying to categorise a set of input patterns.

Lippmann (1987) summarises a number of supervised and unsupervised training algorithms, which are expanded on by Hagan *et al.* (1995).

3.2 The Multilayer Perceptron

3.2.1 Features of the Multilayer Perceptron

The multilayer perceptron is the form of neural network used in the experiments presented in this thesis. It was chosen because of its flexibility in representing different concepts. The main features of the multilayer perceptron that lead to its adoption are presented below.

The DTI *Guidelines for Neural Computing* (DTI, 1994a) state that the multilayer perceptron network is “one of the most popular and successful neural network architectures” and that it is “suited to a wide range of applications”. In a study of neural network applications Sharda (1994) found that 41 out of 52 applications involved multilayer perceptrons.

The multilayer perceptron has a layered network architecture with input, hidden and output layers. Each neuron in one layer is connected to every neuron in the following layer, as shown in Figure 3.4. The neurons in the input layer simply pass on input data values, while those in the hidden and output layers process the data, usually using a sigmoid activation function. The use of the hidden layers allows the multilayer perceptron to model non-linear relationships in the data.

Training data for the network can be a mixture of real values and binary values, though the outputs must be in the range $[0,1]$. This restriction in output values does not prohibit the representation of certain functions, but may require some re-scaling of the data.

The training algorithm, known as backpropagation, was developed by Rumelhart *et al.* (1986) and uses supervised learning, where the network is provided with input patterns along with the corresponding target values.

3.2.2 The Functional Form of the Network

The original formulae for the learning algorithm in the multilayer perceptron are presented by Rumelhart *et al.* (1987), expanded on by McClelland & Rumelhart (1988), and are explained in more depth by Smith (1993). Here, a summary of the main mathematical principles will be presented, with some explanation of their meaning.

Figure 3.5 illustrates the architecture of the multilayer perceptron for the purposes of relating elements of the formulae with the structure of the network.

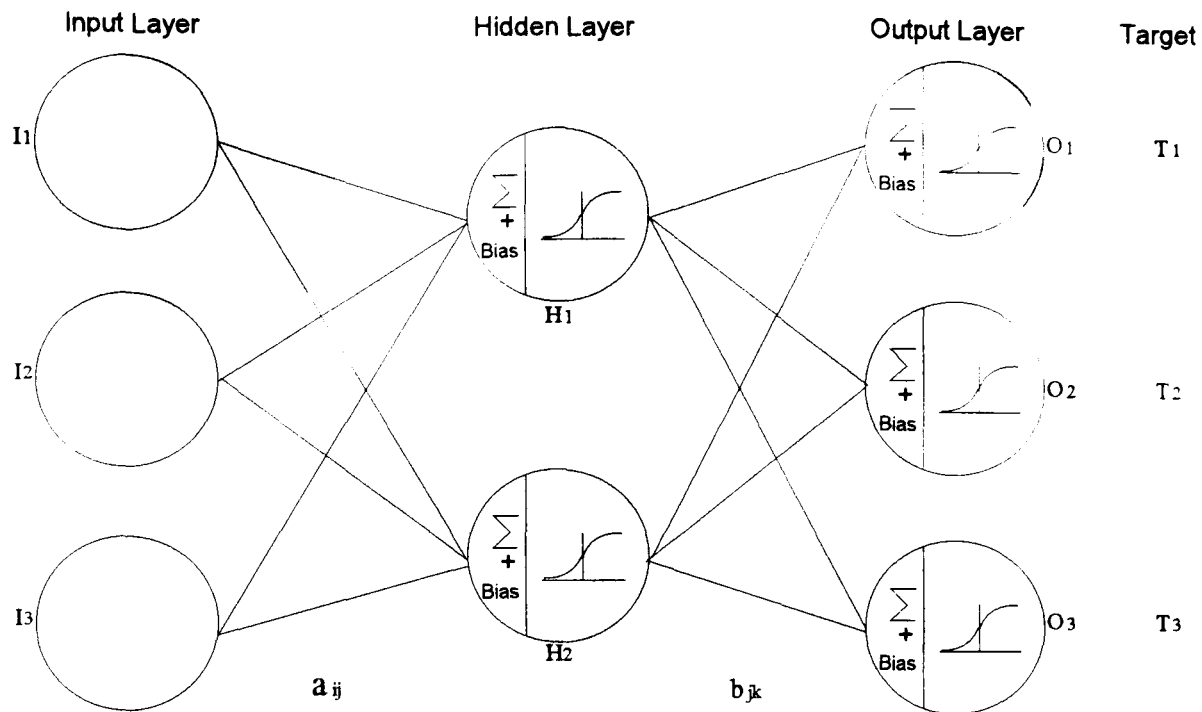


Figure 3.5 : Illustration of a 3*2*3 Multilayer Perceptron

Neuron activation

Firstly, the formula for the activations of the neurons will be examined. The activation of a neuron I_i in the input layer is simply the value of the data point which has been shown to the network. The activations of neurons in the hidden and output layers work in the same way as each other, and so will be demonstrated using the output layer. The structure of these neurons is much as shown in Figure 3.2.

The net sum v_k of input connections to a neuron in the output layer is made up of a bias term $bias_k$ plus the sum of the activations of the hidden neurons H_j , each one multiplied by the connection strength b_{jk} between neurons j and k . Thus,

$$v_k = bias_k + \sum_j b_{jk} H_j$$

The bias term represents the sensitivity of the neuron, while the connection strengths represent the form (positive or negative) and strength of relationship between the neurons.

The net inputs to the neuron are then passed through the sigmoid activation function. Although the shape of the sigmoid function can be achieved using a number different formulae, the most commonly used is the logistic function, which is:

$$O_k(v_k) = \frac{1}{1 + e^{-v_k}}$$

The logistic function gives values in the range $[0, 1]$ and is symmetrical around $v_k=0$.

An input pattern is fed into the neurons in the input layer, and then the response of the network comes from passing the values through the weighted connections and activation functions to the output neurons. This stage is known as the feed-forward phase. On initially setting up the network, the connection values are set to small random values (for example, a uniform distribution from -0.15 to +0.15), but development of the network occurs through changing these connection strengths.

Attributing errors to the weights and biases

After the feed-forward stage, the backpropagation stage occurs. Here the output of each of the output neurons O_k is compared against the target value for the example, T_k , and the square error calculated. The error is then propagated back through the network and attributed to weights and biases. Changes to weights are allocated using the steepest descent method, that is they are changed in the direction that reduces the error most quickly. In most cases the weight and bias changes are stored until all the examples have been shown to the network, and then the net result of all intended changes is applied to the network.

To attribute errors to weights (biases can be treated as a special case of weights) requires the calculation of the partial derivatives of the output error with respect to the weights.

Looking first at the weights b_{jk} which feed into the output layer. The partial derivative of the error E with respect to a weight b is found using the chain rule, as follows:

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial O} \cdot \frac{\partial O}{\partial v} \cdot \frac{\partial v}{\partial b}$$

Here :

- $\partial E / \partial O$ is the derivative of the error with respect to the output value of the neuron,
- $\partial O / \partial v$ is the derivative of the output of the neuron with respect to the sum of its weighted inputs (and the bias term),

- $\partial v / \partial b$ is the derivative of the sum of the weighted inputs with respect to the specific weight.

For a particular output O (dropping the subscript k), the error E compared to the target T is :

$$E = \frac{1}{2}(O - T)^2$$

The $1/2$ is used for computational ease in later calculations, and does not affect the minimisation process. The derivative of E with respect to O is :

$$\frac{\partial E}{\partial O} = O - T$$

The derivative of O with respect to the net of inputs to the node is the derivative of the logistic function. This function has a useful property in that its derivative can be calculated as :

$$\frac{\partial O}{\partial v} = O(1 - O)$$

The derivative of the sum of weights with respect to the weight in question, varies depending on whether it is from another neuron or is the bias term:

$$\frac{\partial v}{\partial b} = \begin{cases} 1 & \text{bias term} \\ H & \text{weight from hidden neuron} \end{cases}$$

Putting the differentials together (and replacing the subscripts) gives :

$$\frac{\partial E_k}{\partial b_{jk}} = \begin{cases} (O_k - T_k)O_k(1 - O_k) & \text{for a bias} \\ (O_k - T_k)O_k(1 - O_k)H_j & \text{for a weight} \end{cases}$$

The process of attributing errors to weights going into the hidden neurons is a similar to that for the output neurons. However, this time the error term is the one for the whole network, that is, the sum of the squared-errors for all of the output neurons. The sum of weighted inputs to a hidden neuron is defined as :

$$u_j = \text{bias}_j + \sum_i a_{ij}I_i$$

Again using the chain rule, the derivative of the network error E with respect to a particular weight going into a hidden neuron, is:

$$\frac{\partial E}{\partial a} = \frac{\partial E}{\partial H} \cdot \frac{\partial H}{\partial u} \cdot \frac{\partial u}{\partial a}$$

Here :

- $\partial E/\partial H$ is the derivative of the network error with respect to the output value of the hidden neuron,
- $\partial H/\partial u$ is the derivative of the output of the neuron with respect to the sum of its weighted inputs (and the bias term),
- $\partial u/\partial a$ is the derivative of the sum of the weighted inputs with respect to the specific weight.

The derivative of the network error with respect to the output of the hidden neuron is the sum of the hidden neuron's contribution to the errors in the output neurons. For all the output neurons ($k = 1..K$) this is :

$$\frac{\partial E}{\partial H} = \sum_{k=1}^K \frac{\partial E_k}{\partial O_k} \cdot \frac{\partial O_k}{\partial v_k} \cdot \frac{\partial v_k}{\partial H}$$

A functional form for the first two components has already been found in the consideration of the error derivatives with respect to the output neurons. $\partial v_k/\partial H$ is the relative contribution of the output of the hidden neuron to the net of weighted inputs to a particular output neuron. This relative contribution is actually the value of the weight between the two neurons, i.e. b_{jk} . Putting the components together for a hidden neuron H_j (dropping subscript j) gives :

$$\frac{\partial E}{\partial H} = \sum_{k=1}^K (O_k - T_k) O_k (1 - O_k) b_k$$

The derivative of the output of the hidden neuron with respect to the net of weighted inputs is, as for the output neurons, down to the logistic function, so :

$$\frac{\partial H}{\partial u} = H(1 - H)$$

Similarly, the derivative of the net of inputs to a hidden neuron with respect to the particular weight, is much as for the output neuron, and varies depending on whether it is a bias term or a weight from an input neuron. This is :

$$\frac{\partial u}{\partial a} = \begin{cases} 1 & \text{bias term} \\ I & \text{weight from input neuron} \end{cases}$$

Therefore, putting all the components together, the contribution of a weight between an input and a hidden neuron is :

$$\frac{\partial E}{\partial a_{ij}} = \begin{cases} \left(\sum_{k=1}^K (O_k - T_k) O_k (1 - O_k) \right) H_j (1 - H_j) & \text{for a bias} \\ \left(\sum_{k=1}^K (O_k - T_k) O_k (1 - O_k) \right) H_j (1 - H_j) I_i & \text{for a weight} \end{cases}$$

While the formulae look relatively complicated, the mathematical operations are simple, and can be accomplished relatively efficiently in a computer program.

Updating weights and biases

For the neural network to adapt, the weights and biases must be changed. The errors on each example pattern are associated with individual weights and biases. Changes to the weights and biases are calculated for each example such that they would reduce the errors. These changes are usually applied when all the examples have been shown to the network. Forming the final network structure is an iterative process, with the examples being repeatedly shown and updates to the network being made.

The sum of the derivatives for a weight over all the examples gives the direction and the magnitude of the error derivative, so the sum of derivatives d_l at iteration l , is :

$$d_l = \sum_{n=1}^N \left(\frac{\partial E}{\partial w_l} \right)_n$$

where n is the index of the example from 1..N in the data set, and w_l is the value of the weight at iteration l . The actual change to the weight, Δw_l , is a proportion ε (known as the gain term) of the error derivative with the sign reversed, so that the error goes down rather than up, so :

$$\Delta w_l = -\varepsilon d_l$$

The value of ε controls the size of the steps in weight changes, and will usually be set in the range (0,1]. The choice of ε can have a large impact on network learning, since

choosing a value which is too large can lead to features on the error surface (including the optimum) being missed, while a value which is too small can result in long network training times. The optimum value of ε depends on the nature of the data, so trial and error is required. An alternative is to use an adaptive approach, as suggested by Vogl *et al.* (1988), where the gain term is allowed to vary depending on the state of the network learning. This adaptive approach will be discussed in more detail in Section 3.3.3.

Another factor which is usually used in making weight changes is a momentum term. This is an exponentially weighted average of the previous changes for a weight. It is used to smooth the training process and reduce oscillation in weight changes. Incorporating a momentum parameter m into the weight changes gives :

$$\Delta w_l = m\Delta w_{l-1} - (1 - m)\varepsilon d_l$$

The momentum parameter can take a value in the range $[0,1)$ where 0 means it has no effect on weight changes. The performance of the network is not highly sensitive to the value for the momentum. The suggested value by McClelland & Rumelhart (1988) for the momentum is 0.9, which Swingler (1996) notes is now the generally accepted value.

3.2.3 The Multilayer Perceptron as a Statistical Tool

Smith (1993) emphasises the statistical aspects of the multilayer perceptron (referring to it by the name of backpropagation), stating :

“Even though backpropagation has played a key role in revitalizing the field of Neural Networks, it is really a statistical modeling technique. More specifically, backpropagation is a non-parametric modeling method: one in which the shape of the relationship between inputs and outputs is decided by the data rather than predetermined by the tool.”

Cheng & Titterton (1994) look at neural networks from a statistical perspective, and draw strong similarities between neural networks and statistical models, pointing out useful links with statistical methodologies.

This emphasis of the statistical aspects of the multilayer perceptron is an important one for thinking about its use in Operational Research applications. The neural network need not be considered a model of the brain but an optimising technique, training examples are no longer stimuli but data that needs to be fitted. Issues of data integrity, explanatory power, generalisation and post-analysis become as important for neural networks as they are for any other statistical modelling tool.

A look at some of the applications for which the multilayer perceptron has been used backs up the view that it can be considered as a statistical tool.

In the area of classification, Hart (1992) did a study of multilayer perceptrons on a number of data sets. This was followed up and expanded on by Curram & Mingers (1994) in a comparative study against linear discriminant analysis and rule induction. DTI (1993) describes applications for credit risk analysis where likelihood of bad debt is assessed, and fruit grading to classify the quality of fruit from analysis of video images. Gorr *et al.* (1994) and Hardgrave *et al.* (1994) both compared the use of neural networks against traditional techniques for predicting student grades. DTI (1994b) reports the use of neural networks by Radio Rentals for improving the effectiveness of their direct mailing campaign through analysis of customer profiles.

For regression modelling, Hoptroff (1993) presents a number of examples of applications using a multilayer perceptron to demonstrate the sorts of problems which can be tackled, with particular emphasis on sales forecasting. DTI (1993) describes a network for predicting television audience numbers for particular programs at particular times. DTI (1995a) gives an account of consumer demand forecasts by Britvic, looking at factors such as unemployment, promotions and the weather. DTI (1995b) gives a brief overview of the use of neural networks in the financial sector. Church and Curram (1996) forecast consumers' expenditure using a variety of econometric variables, comparing the neural network models against more traditional econometric approaches. The testing period for the models was the period from the late 1980s to the early 1990s where existing models of the UK economy had failed to predict the downturn in the economy.

In time series forecasting, Refenes *et al.* (1993) describe experiments in currency exchange rate prediction. Maasoumi *et al.* (1994) used a multilayer perceptron for time series modelling on 14 different data series, while Azoff (1994) devotes a whole book to neural network time series forecasting of financial markets.

The multilayer perceptron has also been used for efficiency and performance assessments between different business units. DTI (1995c) reports case studies for the Sears retails group and Marks & Spencer in forecasting sales volumes for proposed locations of new stores. Athanassopoulos & Curram (1996) compared the use of neural networks and data envelopment analysis (DEA) for efficiency assessment, using both artificial data (with a known level of inefficiency) and actual data of bank branch performances.

3.3 The Practice of Using the Multilayer Perceptron

The previous section looked at the functionality of the multilayer perceptron, and argued for thinking of the approach as one of statistical modelling. Experience with using neural networks in practice has shown that the ways in which the problem situation is conceptualised, the training data is prepared, and the outputs of the models analysed, generally have a very great impact on the quality of the results at the end. The idea that numbers can be pushed into the network at one end, and results churned out at the other without any great thought required by the user is simply untrue. Despite this, no strong methodology for the use of neural networks has developed. This section looks at the lessons for the practical use of the multilayer perceptron with particular emphasis on considering it as a type of statistical tool.

3.3.1 Selecting Variables

The data that is used to train the neural network can be thought of as being made up of explanatory and dependent variables.

As in any data driven approach, the quality of the model is only as good as the data used to develop it. Thus care should be taken to identify key explanatory variables.

Hopcroft (1993) claims that neural networks can extract features from important explanatory variables and ignore variables that do not contribute. This means that a modeller can use what Hopcroft calls 'maybe' variables - those which may possibly provide useful information. Swingler (1996) does not refute this view, but notes that a greater number of input variables needs more data to effectively train the network, and involves more weights, thus increasing training times. Therefore Swingler recommends keeping the number of explanatory variables to a minimum and choosing the most explicit representations of features in the data.

Church & Curram (1996) performed experiments with econometric data, using the same variables as a number of different models in the economy. In the final experiment, they combined the variables from all the models. The combined model outperformed all of the individual models on the test data, but it was found that due to the limited amount of data, the training process became more difficult, and great care had to be taken to choose representative data for the independent validation set (discussed in 3.3.4). It is likely that the training set contained some largely redundant variables, but it does appear that the network was able to cope with these. On the other hand, the increased dimensionality of the model compared to the number of data points made the training process more sensitive to gaps in the training data.

3.3.2 Coding the Variables

The multilayer perceptron can cope with continuous valued and classification variables. Some thought is required on how to turn real life situations into suitable data formats.

Continuous Variables

Continuous valued variables need to be scaled for use as output variables. The logistic activation function can only produce results between 0 and 1, so all output variables need to be scaled to this range. In many cases, it is useful to scale within that range. Both Smith (1993) and Swingler (1996) recommend a range between 0.1 and 0.9. Network training is more efficient on the more linear part of the logistic function in the output node since the differential of the an output (O) with respect to the net inputs to the neuron (v) is :

$$\frac{\partial O}{\partial v} = O(1 - O)$$

This means that as the output approaches 0 or 1, the differential gets close to 0, resulting in extremely small errors being attributed throughout the network. An additional benefit is that it leaves room for extrapolation by the finished network outside the range of the training data, although, as in any statistical approach, care must be taken in interpreting any extrapolated results.

In theory, input variables do not need to be scaled since the weights leading to the hidden layer will adapt and do the job. However, Hart (1992) suggests that input variables should all be scaled to the same order of magnitude to prevent variables with small values being swamped by variables that take much larger values. Typically the range for scaling would be 0 to 1, as for output variables.

Categorical Variables

Categorical variables represent situations where there is no concept of measurable distance between values. Typically they are used to represent classifications or true/false, present/not present situations.

An individual variable is coded using a binary (0 or 1) representation. Where there are several categories of classifications, a more complex representation involving several variables is required.

For output variables it is best to represent each class using a separate variable, with only one switched on at any particular time (e.g. for 3 classes you could use 1 0 0, 0 1 0, and 0 0 1). This has the advantage of being easy to interpret and also, in some situations, gives a degree of membership to each class where there is uncertainty. Swinger (1996) suggests that scaling within the range 0 to 1 (say 0.1 to 0.9) does not work well with categorical data.

For input variables, there is the choice of using same method as for outputs, which has the advantage of clarity, or to represent the classes as binary numbers (e.g. for 3

classes you could use 0 1, 1 0, 1 1) which has the effect of reducing the number of variables required. The use of binary has the drawback of increasing the complexity of representations within the network since it requires interactions between the variables to represent certain classes. This is likely to increase the need for more hidden neurons and so increase training times. Therefore, despite requiring more inputs, the one variable per class method is used more often, although as Smith (1993) points out, one variable can be dropped by representing one of the classes as all the variables being switched off.

Integer Variables

Integer variables offer a choice of being represented as continuous or categorical variables. Swingler (1996) suggests that categories can be considered in cases where the range of possible values is small (say 0, 1, 2 or 3), particularly for outputs where there may be advantages in representing degree of membership. Otherwise a continuous representation can be used.

Dummy Variables

Binaries can be used as dummy variables to represent unusual events or certain time dependent features. Church & Curram (1996) used dummy variables in both the neural network and econometric models of consumers' expenditure to represent periods in which UK value added tax levels changed. Smith (1993) suggests that dummies can be used to represent periodic variation (e.g. seasonality) by using a variable to represent each period.

3.3.3 Setting Gain and Momentum Parameters

As described in Section 3.2.2, the gain and momentum parameters are involved in determining the size of weight updates. The gain represents the size of steps in weight changes, while the momentum specifies the importance of the exponentially weighted average of past weight updates which is used to smooth the learning process.

Swingler (1996) suggests that a value for the gain of 0.25 and for the momentum of 0.9 is generally a good starting point. McClelland & Rumelhart (1988) note that

training can be sensitive to the choice of the gain value, but is less sensitive to the momentum term. Therefore, some experimentation is required in setting the gain term for a particular data set.

An alternative approach suggested by Vogl *et al.* (1988) is to use an algorithm that changes the gain term depending on the current state of the training. The algorithm is as follows :

1. If there is an improvement in the mean squared-error, the gain term is increased, representing increased confidence in the direction of learning.
2. If the mean squared-error worsens by more than a small percentage, this represents a poor direction of learning, so the last weight changes are discarded, the gain term reduced, and the momentum switched off so that past weight changes are not allowed to affect the current changes. The momentum term is switched on after a successful training iteration.
3. Otherwise keep the parameters the same.

Using the algorithm, Vogl *et al.* suggest that training times should be significantly reduced (more than halved in the example with their data).

3.3.4 Generalisability of Results

The multilayer perceptron has the ability to fit different levels of non-linear relationships depending on the number of neurons in the hidden layer. The danger is that it is possible for a network with enough hidden neurons to overfit to the training data, especially where there is noise. The data that is used for training is fitted very accurately but in doing this the network does not capture the true underlying trend. When the network is used to produce results with new data with which it has not been trained, it is unable to generalise the model and so produces poor results.

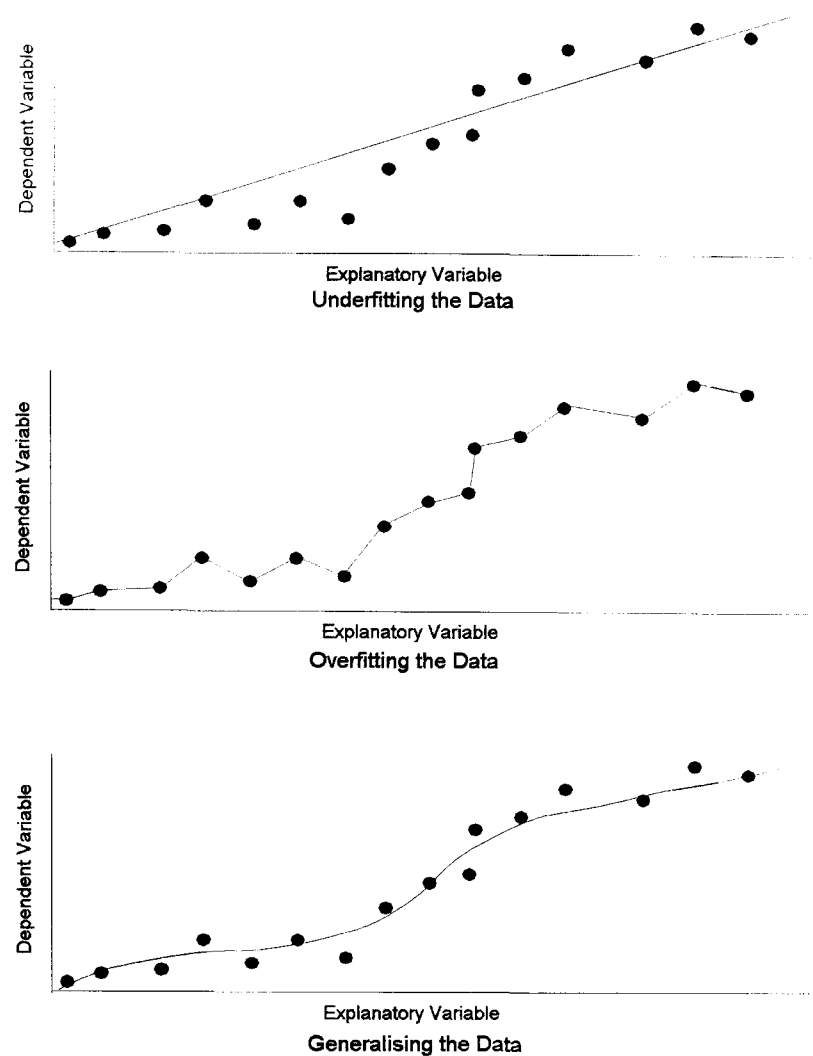


Figure 3.6 : An Illustration of Underfitting, Overfitting and Generalisation

Figure 3.6 shows an example (in a two dimensional case) of underfitting, overfitting and good generalisation in noisy data. In the underfitting case, too few hidden neurons are being used and so the neural network has not got enough power to represent the non-linear relationships in the data. In the overfitting case, there are too many hidden neurons, so the network has fitted the noise in the data, leading to poor generalisation of the underlying structure. In the generalisation case, the neural network has sufficient neurons to fit the underlying structure, but not so many that it goes on to fit the noise.

Determining the Number of Hidden Neurons

The usual approach to trying to optimise the number of hidden nodes is to train the network with a number of different configurations, and measure the mean square-error for each one using a separate test data set. From the starting number of hidden neurons, the network can be retrained using one more and one less hidden neuron, and

the change in the error examined, the direction which leads to an improvement in the error can then be investigated further until the minimum error is found for the test set.

Choosing an appropriate starting point for the number of hidden neurons can speed up the process. Hecht-Neilson (1990) uses Kolmogorov's theorem to suggest that the number of hidden units should not need to be more than twice the number of input variables. Swingler (1996) suggests that for good generalisation, the number of hidden neurons should often be considerably less than this. These are useful as starting guidelines, although they do not always hold in practice, particularly where there are a small number of input variables. There is a certain element of a black art to the choice of hidden neurons, often using past experience of the best choice for data sets with similar features.

As well as being able to vary the number of nodes in a hidden layer, it is also possible to vary the number of hidden layers. Lippmann (1987) shows the difference between using one and two hidden layers on a hypothetical classification problem. In two dimensions, a network with one hidden layer can discriminate in an open or convex region of the two dimensional space. Using two hidden layers allows discrimination in concave or several distinct regions.

For most practical problems one hidden layer is sufficient, particularly where the problem is multi-dimensional. Hornik & White (1989) showed that a network with one hidden layer is a universal approximator for any continuous function. Using more than one hidden layer gives the ability to model very complex relationships but causes problems in trying to develop a model that can generalise for new cases.

A number of approaches have been suggested to automate the process of adding and removing hidden neurons. Swingler (1996) gives an overview of such approaches, but states that none have been shown to be reliable with a large number of data sets, and so the area should be seen as still being under development.

Independent Validation

An alternative to optimising the number of hidden neurons is to stop the network training early before it has had a chance to overfit, by using independent validation. This approach is described by Hoptroff (1993).

The network is required to have sufficient neurons to be able to fit the trend in the data but is not streamlined to prevent overfitting. Hoptroff suggests that 10 neurons in one hidden layer are sufficient for most business and forecasting problems. More nodes can be used but usually result in slower training times without improving the results.

An independent validation data set is used to test how well the network is able to generalise to new data. This validation is separate from the training data - it is used to evaluate the network performance, but is not used in any way for determining weight changes. To be effective, the validation data should be representative across the range of possible outcomes. A large validation set is more likely to be representative than a small one, but in most cases the amount of available data is limited and so would impact the amount of data available for training. It is therefore necessary to strike a balance between the size of the training and validation data sets. Hoptroff (1993) suggests that the validation data should comprise 10-25% of the available data, and have at least 10 data points.

Church & Curram (1996) did experiments with small data sets, but found that using 30% of the data for validation was more appropriate. Since the amount of validation data was very limited, a stratified rather than a random approach was more useful as it increased the likelihood of a representative validation set. In the stratified approach, the data was ordered by the size of the dependent variable (the model having only one output) and the data partitioned into groups of roughly equal size, one for each validation point required. One point was randomly selected from each group to go into the validation data set. One restriction on the algorithm was that the smallest and the largest outputs in the data set could not be selected for the validation data, since it was considered that these boundary values were more important for the training data set.

Each iteration of the training process with independent validation is as follows. The network is presented with the set of training examples, and weight and bias updates are made. The network is then presented with the independent validation set, and the mean squared-error (MSE) is calculated. Training is stopped at the iteration where the MSE of the validation data is minimised since that represents the point where the network has the best generalising ability. In practice, Church & Curram (1996) found that the MSE for the validation set can worsen, and then improve again, so they stored the state of the network at the optimum point, and allowed the network to continue training to ensure that no better value was reached.

Smith (1993) notes that in some cases the MSE of the validation set can continually improve, suggesting that too few neurons are being used in the hidden layer, and so training should be restarted with more hidden neurons.

The reasoning behind the approach given by Hoptroff (1993) is that the underlying trend lies closer to the network state at the start of training than a model which has been overfitted to noise. The gradient descent approach, which is used in the backpropagation algorithm, tries to minimise the fitting error in as short a distance from the starting point as possible. Therefore, it will fit the trend before going on to fit the noise. So, stopping training at the right time prevents overfitting from occurring. Since the validation data is representative and independent of the training data, it should indicate the point where the network generalises best.

Network Regularisation

Swingler (1996) discusses the use of network regularisation, which limits the size that a particular weight can take and so limits the complexity of the network. Two approaches are described, both of which amount to having essentially the same effect.

One approach is to use a regularisation term in the calculation of the network error which adds a proportion of the weights in the network to the mean squared-error calculation for the fitting error. Therefore, the larger the weights become, the more

they contribute to the network error, and so this will act to control the weight sizes. The regularisation error for iteration l is calculated using all Q weight values (w_q) in the network:

$$reg_error_l = \frac{\lambda}{p} \sum_{q=1}^Q w_q^2$$

The extra parameters are p which is the number of training examples (to give a mean error per example) and λ which is a regularisation scale factor that represents the importance of the weights in the error calculation. The regularisation parameter needs to be optimised by experimentation.

An alternative approach is to add noise to the data (even if the data naturally contains noise). Sietsma & Dow (1991) conducted experiments which found that classification rates improved on noisy test data when random noise was added to the training data. Bishop (1995) found that adding noise to the training data was equivalent to using a regularisation term in the error. With added noise, a regularisation parameter λ needs to be set in adding the noise, but this can be varied during the training process. Swingler (1996) recommends starting with a high value for λ of 0.1, using independent validation data to find the best result, and then reducing λ in steps until no improvement is seen on the error with the validation data.

3.3.5 Analysing Neural Network Results

The model formed by training a neural network is embodied in the connection strengths and the biases in the network. Although these could be converted into a straight mathematical function, the result from any but the smallest network would be messy and convoluted.

In most cases the trained neural network is used directly to produce results for new data. This means that when the network is used, the input data needs to be suitably scaled (using the same functions as the original training data), it is presented to the network and a feed-forward step is conducted to produce the network response. If the targets in the training data needed to be scaled, then the network response needs to be re-scaled using the inverse of the scaling function so as to convert the results back to

the original magnitude. Since the network is only being used to provide answers, and is not being trained, no target values are required and a back-propagation step is not performed.

Model Validation

It is important to examine the performance of the network with a test data set, that is one which contains both inputs and targets but which was not used in the training process (i.e. not training data or validation data).

Calculation of the mean squared-error of the test data provides information on how well the network generalises to new data compared with the training data, and so can be used to examine overfitting.

As with other statistical techniques, an examination of the residuals or of classification rates can yield information on areas of the data set where the network may be fitting well or badly. Residual plots can be produced, or Swingler (1996) recommends the use of error histograms to show the pattern of fitting errors. While the neural network is a non-parametric technique, a healthy network should have residuals which are symmetrically spread around 0, and Normally distributed where the noise in the data is Normally distributed.

The network should also be tested with extreme values to see how well it copes with these. Extremes are often not well represented in the training data, so it is useful to discover the bounds within which the network can produce reliable results.

Deriving Explanations from Neural Networks

It is desirable to be able to derive explanations from the network about the relationships it has modelled for reasons of understanding more about the system, and also for justifying the use of the model to the problem owner.

Unlike expert systems, the neural networks do not provide explicit rules for modelling relationships. Rather, the relationships are bound up in the weights and biases spread

throughout the network. It is difficult to derive explanations from this network structure. Wildberger (1990) notes that this difficulty arises because “the relationships embodied in the trained network are only meaningful in a global sense, yet all of the pertinent weights and activation functions are stored locally”.

Wildberger (1990) suggests an approach of tracing the weights between each input and output neuron. The significance S_{ik} of input i on output k is:

$$S_{ik} = Input_i * \sum_j (w_{ij} * w_{jk})$$

where $Input_i$ is the value of the input (this could be set as a constant), w_{ij} is the weight between input neuron i and hidden neuron j , and w_{jk} is the weight between the hidden node j and the output k . Wildberger says that these should be viewed more as a qualitative representation of the importance of each input, rather than be taken in their strict quantitative sense. The approach can be thought of as converting the weight matrix into a crude linear model. The weakness of the approach is that it takes no account of the effects of neuron biases, and takes no account of non-linearities or interactions between inputs.

Church & Curram (1996) used a perturbation approach for sensitivity analysis which is borrowed from econometric modelling. Here all of the variables were set to their average values, and the value of one variable in turn was increased (shocked) by 1%. The response of this change was then calculated in terms of the change in the dependent variable. This approach was compared against the elasticities of an econometric model based on the same data, and it was found that both had similar response structures. This approach assumes that the variables have linear structures so as to be generalisable across all values for the variables, otherwise the approach can only be seen as local, giving an idea of response to small changes from the average. Church and Curram used an extension of the perturbation approach where each variable was altered in 1% steps between -20% and +20% from its average value, while all other variables were kept the same. This approach gave an indication in the degree of non-linearity for each variable, although it did not test the interaction effects between variables.

It would be possible to test the interactions between variables using the perturbation approach, by increasing several variables together to find the effect on the dependent variables. However, the number of combinations to test would increase dramatically with the number of input variables.

Therefore it is possible to derive an idea of the relationships in the model but this is better done through the analysis of the inputs and outputs, rather than the structure of the network weights.

3.4 Neural Network Representations of Stochastic Processes

The concept of using neural networks to represent variability in intelligent decision making is a new one. Other research has suggested ways in which neural networks can represent stochastic elements in other problem areas. These approaches are considered and developed to suggest ways in which stochastic processes might be introduced to decision making.

3.4.1 The Characteristics of Stochastic Neural Network Representations

A key desirable characteristic of the neural network approach for representing intelligent decision making is the potential for incorporating variability in decision making. This variability can be treated as residual (i.e. unexplained) variation, which is likely to be caused either by criteria in the system which have not been identified, or by criteria which are brought in from outside the system.

The standard approach used in simulation to represent variability is sampling from probability distributions. Extending this approach to representing decision making, the requirement would be for a neural network model to capture as much of the variation explained by factors within the system as possible, and represent the rest of the variation as a probability distribution which can be sampled from.

Two types of decision representation are considered, which differ in terms of the nature of the decision outcome. Both forms can be modelled with a multilayer

perceptron, but will involve data with different characteristics. One representation is where the decision outcome is a point on a continuum of possibilities, such as a control variable or time. The other representation is where there are a set of options that can be taken which are mutually exclusive. The representation is that of a classification where each decision can be placed into one of a finite set of options.

Both types of decision, when not trying to represent stochastic variability, would be treated differently, one as a regression task, the other as a classification task. Since the starting points differ, they will also be treated as separate problems when trying to represent stochastic variability.

3.4.2 Real-Valued Decision Outcomes

This type of decision involves one or more real-valued outputs in response to a particular set of circumstances. When not allowing for any variability in the decision making, this would be treated as a regression problem, where the behaviour would be a function of the decision criteria.

If the variation in decisions could be assumed to have a standard theoretical distribution which is of the same type across all the circumstances then the outputs of the neural network could just be in the form of the parameters for that distribution. For instance, if the decision is in the form of a single control variable, and the variation in decisions is thought to follow a normal distribution, then the results of the network would be a mean and standard deviation. The distribution would then be sampled from using the parameters given by the network to find the decision. Such an approach is likely to involve a number of assumptions about the distribution and a great deal of processing of the data to determine the parameters for the range of circumstances. Validating that the distribution is appropriate over the range of circumstances is likely to be difficult because of the size of the task.

The more common situation is that a theoretical distribution for the variability cannot be assumed, so the neural network itself must take on the characteristics of the variability.

Hurion (1993) investigated the use of neural networks for representing probability distributions. In this work, the inverse cumulative distribution functions were being modelled. The networks were trained on theoretical distributions with fixed parameters and on empirical distributions. In each case, the output represented a time duration while the input was a cumulative probability.

Hurion used the same training process for both the theoretical and empirical data sets, the difference between the two being the source of the data. The approach used is as follows. The n data points are sorted in ascending order, and cumulative probabilities, p_i are assigned based on the position i in the list using the formula: $p_i = (i-0.5)/n-1$. The neural network is trained with the p_i 's as inputs, with the data values as the corresponding outputs. Hurion reports that a multilayer perceptron was used with a 1-3-3-1 architecture (1 input, 2 hidden layers with 3 neurons each, and one output), after trying a variety of architectures from 1-20-20-1 down to 1-2-2-1. However, no networks with only one hidden layer were tried. The size of the data sets ranged from between about 60 to 140 examples. The results of the investigation showed that the neural network models provided good fits to the distributions, and interpolated well between the training data points.

In the case of modelling variability in intelligent decision making, the variables representing the decision criteria can be included in the model, with an extra variable to represent the unexplained variation, much in the same way as Hurion uses the probability variable in representing distributions.

In setting up the training data, the cases where the decision criteria are identical or very close to each other would be grouped together. The differences between the outcomes would then be allocated to cumulative probability values by the ranked position using the same approach as Hurion (1993). In some cases, it might be desirable to impose artificial extreme points for the variation if it is felt that the data collected does not represent the extreme possibilities. The trained network would be used by sampling a

random number and applying this to the network, along with the decision criteria, to produce a response.

This approach would require the network to be able to interpolate between the decision criteria and the probability values it was trained on, in order to generalise the variable decision behaviour. There would also need to be an amount of pre-processing of the data to form the training set. While it should be possible to do this with one output variable, provided sufficient data is available, it would be more difficult to assign probabilities where there are several output variables. One possibility is to use a probability variable for each output, although this would then assume that the residual variation for the output variables are independent of each other.

3.4.3 Classification Type Decision Outcomes

It is a classification task when the decision possibilities are a set of mutually exclusive classes. The standard approach would be to represent each class as a binary variable, so that for each example only one of the outputs is activated, representing the chosen decision class. A number of options exist for representing the variability in the decision classes, which use this binary representation to varying degrees.

Using a Probability Variable

An approach similar to that for continuous valued outputs could be used, where an extra input variable represents the unexplained variation as a probability distribution. As for the continuous case, the examples with identical or very similar decision criteria would be grouped together, and analysed to empirically determine the probability distribution. The difference would be that the outcome variables change in steps in response to the inputs, i.e. for a set of probability values the result would be one class, and then at a particular point in the probability values the outcome class would be changed. When using the trained network, a random number would be applied to the network, along with the decision criteria to return a response. A winner takes all approach would be used with the output responses, taking the class which has the highest output value.

This approach would require the network to learn a step function to model the sudden jumps in the output values, and to be able to interpolate this across the decision criteria variables. This calls for being able to model a fair degree of non-linearity.

Representing Class Membership Probabilities using Processed Training Data

Rather than using a probability term in the model, the training data could be analysed to determine the probability of class membership for a particular set of decision circumstances. Here, as in the previous method, the examples with similar decision criteria would be grouped together, and the proportion (probability) of those examples falling in each class would be determined. The network would then be trained using targets that conformed to these probabilities, rather than binary values.

For instance, the following is part of a data set and represents a number of cases where different people were faced with the same decision circumstances (inputs) and differing decision outcome classes were observed :

Input 1	Input 2	Input 3	Class A	Class B	Class C
0.5	1	0.4	1	0	0
0.5	1	0.4	1	0	0
0.5	1	0.4	0	1	0
0.5	1	0.4	0	1	0
0.5	1	0.4	0	1	0
0.5	1	0.4	0	1	0
0.5	1	0.4	0	1	0
0.5	1	0.4	0	1	0
0.5	1	0.4	0	1	0
0.5	1	0.4	0	0	1

By calculating the relative frequencies of the different decisions it is found that in this case 20% of people made decision A, 70% decision B and 10% decision C. This would then form one training example of:

Input 1	Input 2	Input 3	Class A	Class B	Class C
0.5	1	0.4	0.2	0.7	0.1

In the trained network, the network output would be the class probabilities, and then a random number would need to be sampled to determine which decision was made.

This approach requires the network to interpolate across the input values for the change in class probabilities. An advantage over the previous approach is that a number of data points from the raw data have been combined together in forming the training set. This means that the training set will be smaller, so that network training time per iteration will be quicker.

Allowing the Network to Determine Class Probabilities

Richard & Lippmann (1992) show that the outcomes of neural network classifications can be considered to be the probabilities of class membership. Smith (1993) summarises the conditions for this as:

1. The network has enough hidden neurons to accurately represent the relationships in the data,
2. The training data set is large enough to prevent overfitting without limiting training time,
3. The training sample is representative of the population
4. There is an output node for each class, and in each example only one has a value of 1 and the rest 0.

This implies that early stopping using a validation data set should not be used in network training. Therefore it is necessary to find the most appropriate number of neurons in the hidden layer, though having larger amounts of data available reduces the sensitivity to this. Smith (1993) notes that in the sum of the probabilities of the classes should be 1, so it may be necessary to re-scale the outputs of the trained network slightly to ensure this.

DTI (1994a) notes that it is often difficult to balance the training set so that class frequencies are representative of the population, and suggests that the outputs of the trained network can be scaled produce more accurate probabilities using :

$$A = (\text{Network Output}) * \frac{(\text{Actual Class Probability})}{(\text{Fraction of Class in Training Set})}$$

$$B = (1 - \text{Network Output}) * \frac{(1 - \text{Actual Class Probability})}{(1 - \text{Fraction of Class in Training Set})}$$

The actual probability of the class is :

$$P = \frac{A}{(A + B)}$$

Swingler (1996) notes that the formula given by the DTI gets increasingly sensitive to errors in the network output as the difference between the training set class frequencies and the population frequencies increase.

In representing intelligent decision making, the network training process can be used to assign the class probabilities. The training data would contain binary decision outputs, but the outcomes from the trained network would be treated as the probabilities of class membership. Since the population class frequencies are unlikely to be known in practice, these must be estimated from the data. Therefore, if some of the data is to be used for testing, it must be ensured that the split of data between training and testing should not effect the class frequencies in either data set. This means that the re-scaling suggested by DTI (1994a) should not be required.

The advantage of the approach is that it does not require pre-processing of the training data to determine the probabilities, leaving it up to the network to determine these. This should prevent having to do extra work to the data prior to training. However, early stopping using validation cannot be used, so optimisation of the hidden neurons is required.

3.4.4 Overview of Stochastic Representations

A number of alternative approaches for representing variability in decision making have been discussed. All of the approaches have particular advantages and disadvantages, but all use the same sort of data for determining the model. The main differences in the approaches are the processing of the data required to produce the training data set, and the ways that the results from the final trained network are used. However, the key criteria for choosing which representations to use is the degree of accuracy with which they can model the decision making behaviour and the variability in decisions.

Chapter 4 will describe the investigation of each of the approaches using artificially generated data so the accuracy of the representations can be gauged against known benchmarks.

Chapter 4

Examining Stochastic Neural Network Representations using Artificially Generated Data

This chapter investigates the potential of using neural networks for developing stochastic models of decision making (that is where decisions under the same circumstances can vary due to different judgements by decision makers) using artificially generated data from probability distributions.

Section 1 gives an outline of the approach to be taken in the investigation. Section 2 gives a brief description of the neural network software that was developed for the work. Section 3 concentrates on models where the output is a continuous control variable, while Section 4 concerns classification outputs. Section 5 discusses the use of independent validation data. Section 6 summarises and comments on the findings of the analysis.

4.1 Method of Investigation

Chapter 3 looked at neural networks in detail, with some consideration of approaches for representing probabilities and stochastic processes. A number of possible alternative approaches were suggested, each of which has strengths and weaknesses. However, a key criteria is the accuracy with which the approach can represent the relationships.

The use of artificial data allows a judgement of the accuracy of representation against known functions. Using classical probability distributions allows sources of data with stochastic elements and known behaviour. The task is similar to Hurrion's (1993) for fitting distributions, but here the networks will be trained with variable parameters (such as location, shape etc.) for the distributions rather than training to represent a

distribution with the parameters fixed. This allows an evaluation of the approaches for representation where there are conditional variables.

The networks were trained using data sets of different distributions and different sample sizes. Also alternative methods of sampling from the distributions were used, one being to choose data points in step sizes for the parameters, the other to use more a random method of selecting data points.

In training the networks, different network configurations in terms of the number of nodes in the hidden layer were tried. The use of independent validation is discussed later.

All of the trained networks were tested using a test data set to measure the their ability to capture the behaviour of the distributions. The results will be evaluated on the size of the fitting errors, and an examination of the residuals to look at the structure of the fitting errors.

4.2 The Development of the Neural Network Software

The software used for implementing the multilayer perceptron has been developed using Borland Pascal 7.0 for the PC running in DOS. The following is a brief description of the main features of the software. A more detailed description is given in Appendix A.

The basic backpropagation engine was developed based on a description by McClelland & Rumelhart (1988). It follows the algorithms described in Section 3.2. Most of the code is contained in libraries (Pascal Units) so that a master program can be created which chooses the desired features from the libraries. This was done so that the neural network software could be developed further with relative ease as new ideas came along.

The software is designed so that the networks can be configured at run-time without having to change program code. At the same time, there was a wish not to place

restrictions on network sizes (number of inputs, outputs, hidden nodes), so an approach using linked lists (dynamic memory allocation) was devised to allow maximum flexibility in network sizes.

The main additions to the basic backpropagation algorithm are the use of an adaptive gain term as described by Vogl *et al.* (1988), and an option to use independent validation, based on the approach described by Hoptroff (1993).

The software allows the state of the network to be saved, so that it can be re-loaded once training has been completed allowing it to be used with test data. The results from the network can be viewed on the screen, or written to a file. Additional code was written that allows a run-time version of the trained networks to be used in other Pascal programs. This allows several neural networks to be loaded into memory, and responses from the network can be obtained for use elsewhere in the program.

Data to be used with the network must be scaled before being used with the software. This can be done using spreadsheet or statistical software, or using another special purpose built Pascal program for data preparation.

4.3 Representing Continuous Valued Output Variables

4.3.1 Approach to Representing Decisions with Continuous Valued Outputs

Ideas for representing decisions with continuous valued outputs (such as control variables) were discussed in Section 3.4.2. The most promising was an extension of the approach used by Hurion (1993) where a probability term is added as an input for training the network, so that variation in decision making can be treated like a probability distribution. In Hurion's case, the probability input was the only input to the network, whereas here, it is in addition to the criteria which are used in making the decision. In the case of the experiments described in this chapter, the 'decision' criteria are the parameters of the distribution that is being represented.

The approach involves creating empirical probabilities for each of a set of criteria values. The decision criteria may be in the form of classification or continuous values, but for the approach to work they need to be placed in appropriate classes. The size of the classes is dependent on the number of variables involved and the amount of data that is available.

The data preparation stage is as follows:

1. The data should be organised into sets of examples, with appropriate numerical values for the decision criteria and the output.
2. Sort the data based on the value of the first decision variable. Choose appropriate classes for the first decision variable and allocate each example to a class. The actual values of the first decision variable should be replaced with the appropriate class mid-point values.
3. *Within* each class, repeat *step 2* for each of the other decision variables.
4. The data set should now be sorted into groups which have the same values for each of the decision variables. For each group, sort the output into ascending order.
5. For each group, assign empirical probabilities (p_i) using the formula:

$$p_i = \frac{i - 0.5}{n}$$

where i is the position of the example in the group, and n is the total number of examples within the group.

An example of the data preparation stage follows. The amount of data is considerably less than would be used in practice, but it demonstrates the transformation process.

Stage 1:

The data is organised into examples, by row, and has two inputs and an output. Input 1 is a continuous variable, and Input 2 is a classification variable. The Output variable is continuous.

Input 1	Input 2	Output
0.23	1	0.52
0.32	2	0.21
0.29	1	0.45
0.16	1	0.63
0.24	1	0.41
0.26	2	0.34
0.17	2	0.28
0.34	1	0.71
0.21	2	0.43
0.29	1	0.74
0.33	2	0.51

Stage 2:

The examples are sorted by the value of Input 1. Input 1 is then divided up into two classes with ranges $[0.15, 0.25)$ and $[0.25, 0.35)$, with the examples in each class taking the class mid-point value. The choice of the number of classes is arbitrary here, but usually the decision will be based on the amount of data that is available.

Input 1	Input 2	Output
0.16	1	0.63
0.17	2	0.28
0.21	2	0.43
0.23	1	0.52
0.24	1	0.41
0.26	2	0.34
0.29	1	0.45
0.29	1	0.74
0.32	2	0.21
0.33	2	0.51
0.34	1	0.71

Input 1	Input 2	Output
0.2	1	0.63
0.2	2	0.28
0.2	2	0.43
0.2	1	0.52
0.2	1	0.41
0.3	2	0.34
0.3	1	0.45
0.3	1	0.74
0.3	2	0.21
0.3	2	0.51
0.3	1	0.71

Stage 3:

The sorting process is repeated for Input 2. This has two possible class values, which will be treated as two separate groups. The result is that the data has been divided into four different groups based on the values of the decision inputs.

Input 1	Input 2	Output
0.2	1	0.63
0.2	1	0.52
0.2	1	0.41
0.2	2	0.28
0.2	2	0.43
0.3	1	0.45
0.3	1	0.74
0.3	1	0.71
0.3	2	0.34
0.3	2	0.21
0.3	2	0.51

Stage 4:

Within each of the four groups, the values of the output variable are sorted into ascending order.

Input 1	Input 2	Output
0.2	1	0.41
0.2	1	0.52
0.2	1	0.63
0.2	2	0.28
0.2	2	0.43
0.3	1	0.45
0.3	1	0.71
0.3	1	0.74
0.3	2	0.21
0.3	2	0.34
0.3	2	0.51

Stage 5:

For each group, an input variable (RV) is added that represents the empirical probabilities. For instance, the first probability is $(1-0.5)/3 = 0.1667$.

Input 1	Input 2	RV	Output
0.2	1	0.1667	0.41
0.2	1	0.5000	0.52
0.2	1	0.8333	0.63
0.2	2	0.2500	0.28
0.2	2	0.7500	0.43
0.3	1	0.1667	0.45
0.3	1	0.5000	0.71
0.3	1	0.8333	0.74
0.3	2	0.1667	0.21
0.3	2	0.5000	0.34
0.3	2	0.8333	0.51

The examples that are used in the neural network have to be scaled so that values are in the range (0,1). This scaling can be done separately for each of the variables. Once scaled, the data at the end of Stage 5 is in a position to be used for training the neural network. The network would have 3 input nodes and 1 output node. The number of hidden nodes would depend on the degree of non-linearity that is suitable for fitting the generalised model. Figure 4.1 shows the neural network shape that would be trained using the data in the example, with an unspecified number of nodes in the hidden layer.

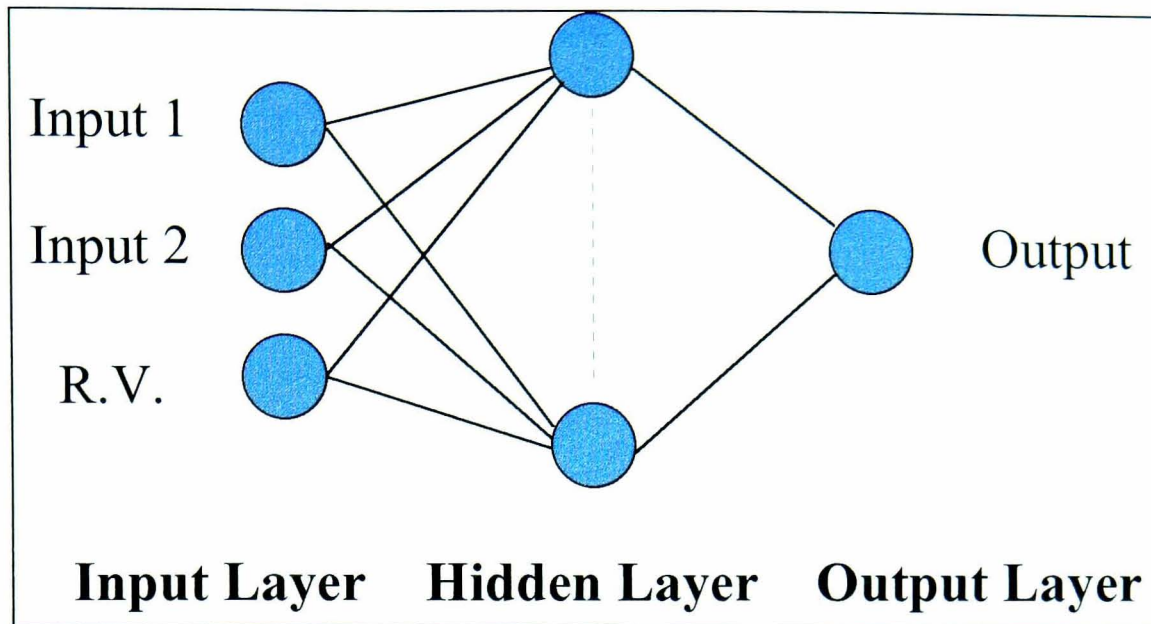


Figure 4.1 : Example Neural Network Structure

4.3.2 Continuous Data Sets

The ability of the neural network to represent distributions of continuous variables is investigated using the Beta distribution, firstly only varying one of the parameters (α_1), and then varying both (α_1 and α_2). The Beta distribution was chosen because of the large variety of shapes it can take, depending on the values of the two parameters. The density function of the Beta distribution is :

$$f(x) = \begin{cases} \frac{x^{\alpha_1-1}(1-x)^{\alpha_2-1}}{B(\alpha_1, \alpha_2)} & \text{if } 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{where } B(z_1, z_2) = \int_0^1 t^{z_1-1}(1-t)^{z_2-1} dt$$

$$\text{for } z_1 > 0 \text{ and } z_2 > 0$$

Figure 4.2, taken from Law & Kelton (1991, p339), shows the variety of shapes that the Beta distribution can take.

Several sets of data were used each containing a different combination of features. These features were: i) varying one or both of the parameters, ii) using different sample sizes, and iii) using known probabilities generated from the inverse cumulative density

function of the Beta distribution, or using a random selection from the distribution with the probabilities calculated empirically from the sample, as explained in Section 4.3.1. The idea of this is that the actual probabilities give the most accurate information on the distribution for the given sample size, against which the data set with the empirical (and therefore estimated) probabilities can be compared.

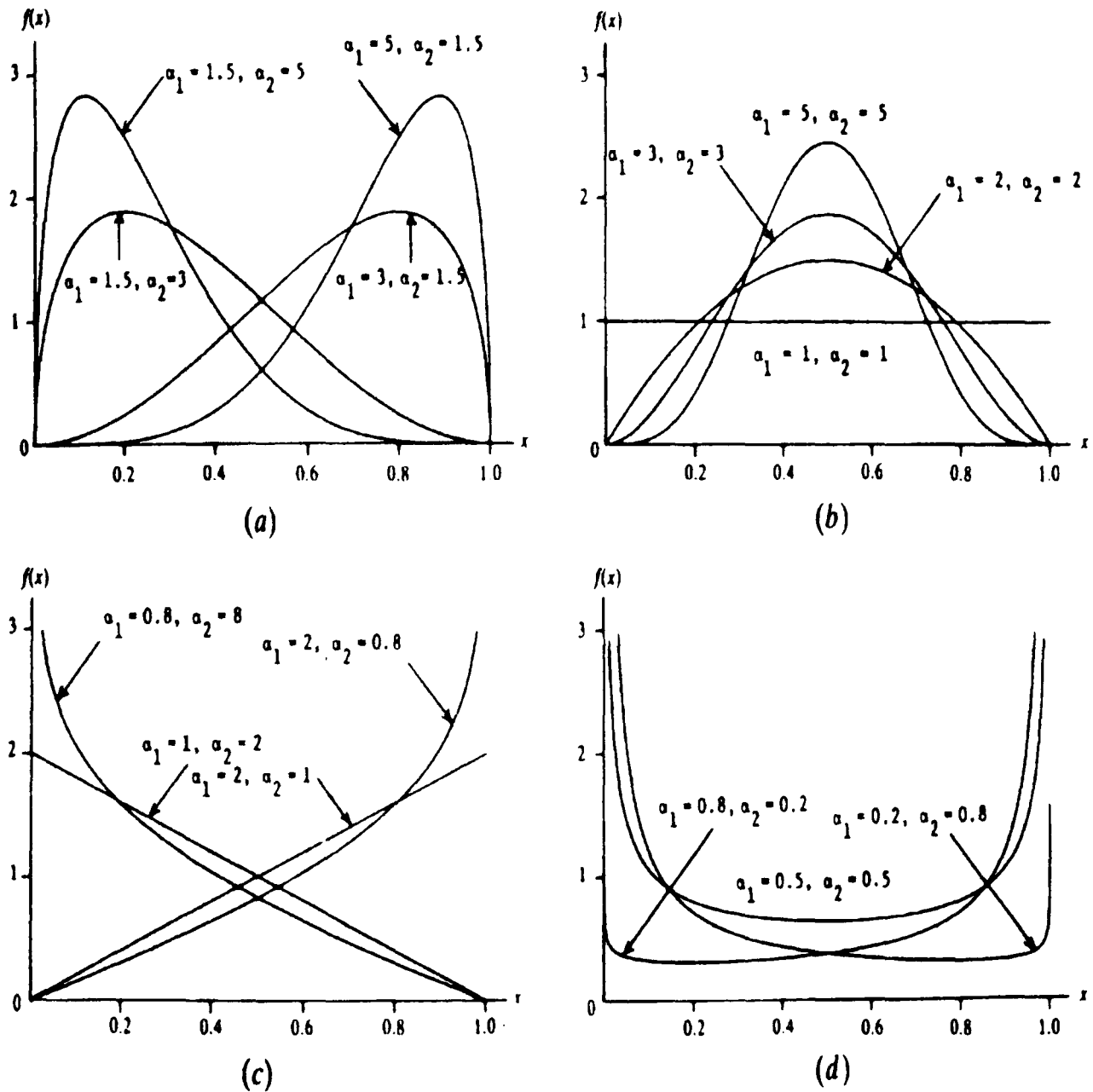


Figure 4.2 : Density Functions for the Beta(α_1, α_2) Distribution

4.3.2.1 Beta Distribution Varying α_1

Data sets with 66 and 231 examples were developed. These sizes were based on the structure of the data sets using known probabilities. The data sets with empirical probabilities used the same amount of data. Table 4.1 shows the values chosen for the

parameter α_1 and the probabilities r , for the structured data set with 66 examples (11*6). These values were used to generate values from the inverse cumulative density function of the Beta($\alpha_1,1$) distribution.

α_1	0.05	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	1.95
r	0.05	0.2	0.4	0.6	0.8	0.95					

Table 4.1 : Input values for Beta($\alpha_1,1$) : 66 examples (Small-Structured)

For the small randomly generated data, 66 values for α_1 were randomly chosen from the range (0,2), with corresponding random probability values. These were used to generate outcome values from the inverse density function of the Beta($\alpha_1,1$) distribution. The probability values that were used to generate the outcome were then no longer used, being replaced by empirical probabilities as determined using the approach outlined in Section 4.3.1. Table 4.2 shows the classifications into which the values of parameter α_1 were placed, with the minimum and maximum values for the classes, and the mid-point of each class. These classes were chosen so that the mid-points matched (except for the extremes) the values used to construct the structured data set.

Min	0	0.1	0.3	0.5	0.7	0.9	1.1	1.3	1.5	1.7	1.9
Mid	0.05	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	1.95
Max	0.099	0.299	0.499	0.699	0.899	1.099	1.299	1.499	1.699	1.899	2.0

Table 4.2 : Ranges and Mid-Points for α_1 : 66 Examples (Small-Random)

A similar process was followed for the data set with 231 examples (21*11). For the structured data set, Table 4.3 shows the chosen values for α_1 and the probabilities r which were used to generate values from inverse cumulative density function of the Beta($\alpha_1,1$) distribution. Table 4.4 shows the classifications used for the randomly generated data with empirical probabilities.

α_1	0.05	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.8	1.0
	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0	
r	0.05	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.95

Table 4.3 : Input values for Beta($\alpha_1,1$) : 231 examples (Large-Structured)

Min	0	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
Mid	0.025	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Max	0.049	0.149	0.249	0.349	0.449	0.549	0.649	0.749	0.849	0.949	1.049
Min	1.05	1.15	1.25	1.35	1.45	1.55	1.65	1.75	1.85	1.95	
Mid	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.975	
Max	1.149	1.249	1.349	1.449	1.549	1.649	1.749	1.849	1.949	2.0	

Table 4.4 : Ranges and Mid-Points for α_1 : 231 Examples (Large-Random)

With the larger data set, while there are twice as many classes as in the smaller data set, there are also, on average, twice as many examples in each class (since the values are randomly generated there is no guarantee that there will be the same number of examples in each class).

4.3.2.2 Beta Distribution Varying α_1 and α_2

Data sets which varied both of the shape parameters (α_1 and α_2) were developed. These data sets used the same degree of sparsity in the examples as used in the case of varying only one parameter, however this necessitates having larger amounts of data due to the extra parameter. The smaller data sets contain 726 ($11*11*6$) data points, while the larger sets contain 4851 ($21*21*11$) data points, again determined from the process of creating the structured data sets. Parameters α_1 and r take the same values as shown in Tables 4.1 and 4.2, while parameter α_2 takes the same values as α_1 for the structured data set. The randomly generated data sets take the same approach and classes as those used for the case where only one parameter was varied. The classes for α_1 (using the same for α_2) and r are shown in Tables 4.3 and 4.4.

In summary, the data sets used were :

Beta($\alpha_1, 1$),	structured values/known probabilities,	66 examples
Beta($\alpha_1, 1$),	random values/empirical probabilities,	66 examples
Beta($\alpha_1, 1$),	structured values/known probabilities,	231 examples
Beta($\alpha_1, 1$),	random values/empirical probabilities,	231 examples
Beta(α_1, α_2),	structured values/known probabilities,	726 examples
Beta(α_1, α_2),	random values/empirical probabilities,	726 examples
Beta(α_1, α_2),	structured values/known probabilities,	4851 examples
Beta(α_1, α_2),	random values/empirical probabilities,	4851 examples

4.3.3 Neural Network Training and Testing

Each of the training sets was used to train networks with a variety of hidden layer sizes. These were : 4, 8 and 12 neurons. In some cases the results of the training indicated that looking at network sizes outside this range might be useful and so this was followed up. All data was scaled to the range (0,1) using a linear scaling approach, scaling from the maximum and minimum possible values (not necessarily those found in the data set). For the inputs, this was done so that all variables had the same order of magnitude. For the outputs it was desired that the output values of the network should be strictly bounded, that is the network outputs cannot go outside the range (0,1) as is the case in the original beta distribution.

In most cases the networks were allowed to continue for 50,000 iterations before being stopped. The network error can in fact rise at certain points during training, so that the networks were actually stopped at the point after 50,000 iterations where the total sum of squares for the training set was minimised. The exception to the use of 50,000 iterations was the case for the $\text{Beta}(\alpha_1, \alpha_2)$ distribution with the large data sets (structured and random) which used 30,000 iterations due to the very long training times involved.

Each of the neural network models was tested with data that had not been used in the training process and so determined the ability of the networks to generalise from the training data to represent the whole distribution. For the each of the $\text{Beta}(\alpha_1, 1)$ and $\text{Beta}(\alpha_1, \alpha_2)$ cases, the test data comprised of 2000 data points that were generated by randomly choosing values in the range (0,2) for α_1 and α_2 (where varied), and (0,1) for r , with true output values generated from the Beta inverse cumulative density function.

Each of the networks was tested with the appropriate test data set to get the network response. These results were then compared against the actual values from the inverse cumulative density function of the Beta distribution to calculate the Mean-Squared Error (MSE) and the Mean Absolute Deviation (MAD).

$$MSE = \frac{1}{n} \sum_{i=1}^n (O_i^{True} - O_i^{NNet})^2$$

$$MAD = \frac{1}{n} \sum_{i=1}^n |O_i^{True} - O_i^{NNet}|$$

where O_i^{True} is the true value from the Beta inverse density function, O_i^{NNet} is the value from the neural network for each point $i = 1 \dots n$ in the test data set.

Through analysing the MAD of the testing sets for each network, the most appropriate size was chosen for each data set, and the network re-trained using 200,000 iterations (or 100,000 for the large Beta(α_1, α_2) data sets) to see how much effect the extra training time had on the results.

4.3.4 Results of Experimentation

The main results and observations from the experiments are given below for the Beta distribution, treating separately the cases of varying one and both parameters.

4.3.4.1 Beta($\alpha_1, 1$) Distribution

A summary of the MSEs and MADs for the experiments are given in Table 4.5. In all cases the error for the training data is considerably lower than for the testing data. The table shows that the average error is lower for larger data sets, due to having more information available so that the network only needs to interpolate over smaller ranges.

The results are considerably worse for the random than the structured data sets. This is hardly surprising since the structured data sets contain accurate information that is evenly spread over the value ranges. The random data sets contain estimated probabilities which assume that the probabilities are evenly spread, although due to the effects of random sampling this is unlikely to be true. The larger data set has the advantage of having a finer grid of classes, and also is likely to have better estimates of the probabilities due to having, on average, twice as many observations in each class.

Small - Structured				Small - Random			
	Training	Testing			Training	Testing	
Size	MSE	MSE	MAD	Size	MSE	MSE	MAD
2*4*1	0.000106	0.000503	0.012767	2*4*1	0.006455	0.032014	0.136740
2*8*1	0.000011	0.000297	0.007193	2*8*1	0.002470	0.045350	0.153603
2*12*1	0.000017	0.000305	0.007716	2*12*1	0.003894	0.042380	0.146971
2*8*1(L)	0.000008	0.000265	0.007386	2*2*1	0.019318	0.017693	0.113350
				2*1*1	0.030455	0.014594	0.095736
				2*1*1(L)	0.030409	0.014671	0.095981
Large - Structured				Large - Random			
	Training	Testing			Training	Testing	
Size	MSE	MSE	MAD	Size	MSE	MSE	MAD
2*4*1	0.000081	0.000198	0.008240	2*4*1	0.009960	0.004870	0.051941
2*8*1	0.000022	0.000076	0.003496	2*8*1	0.005116	0.010681	0.078083
2*12*1	0.000042	0.000123	0.005080	2*12*1	0.006967	0.007604	0.067300
2*16*1	0.000041	0.000123	0.005119	2*2*1	0.015076	0.001366	0.026842
2*8*1(L)	0.000008	0.000047	0.002638	2*1*1	0.020529	0.007560	0.062484
				2*2*1(L)	0.009591	0.004139	0.044741

(L) Refers to network trained for 200,000 iterations

Table 4.5 : Beta($\alpha_1,1$) : Mean Square Errors & Mean Absolute Deviations of the Neural Network Models

Observing the effects of network sizes in Table 4.5 shows that smaller networks are better with the randomly generated data. This is particularly the case for the small training set where results are considerably better for the smaller network sizes. This suggests that overfitting is likely to be more of a problem in data sets where there are fewer data points in relation to the dimensions of the problem.

The networks that gave the best MAD results for each data set were selected for further analysis of the residuals. Figures 4.3 to 4.6 show residual plots for the models of the Beta($\alpha_1,1$) distribution and each of the input variables. This allows an analysis of any structural deficiencies in the models, and the input values that cause most problems. In particular there is a comparison between the residuals of the structured and random data sets. Considering each of the dimensions separately gives an idea of the data regions that are providing most difficulty in fitting the distribution using the neural network.

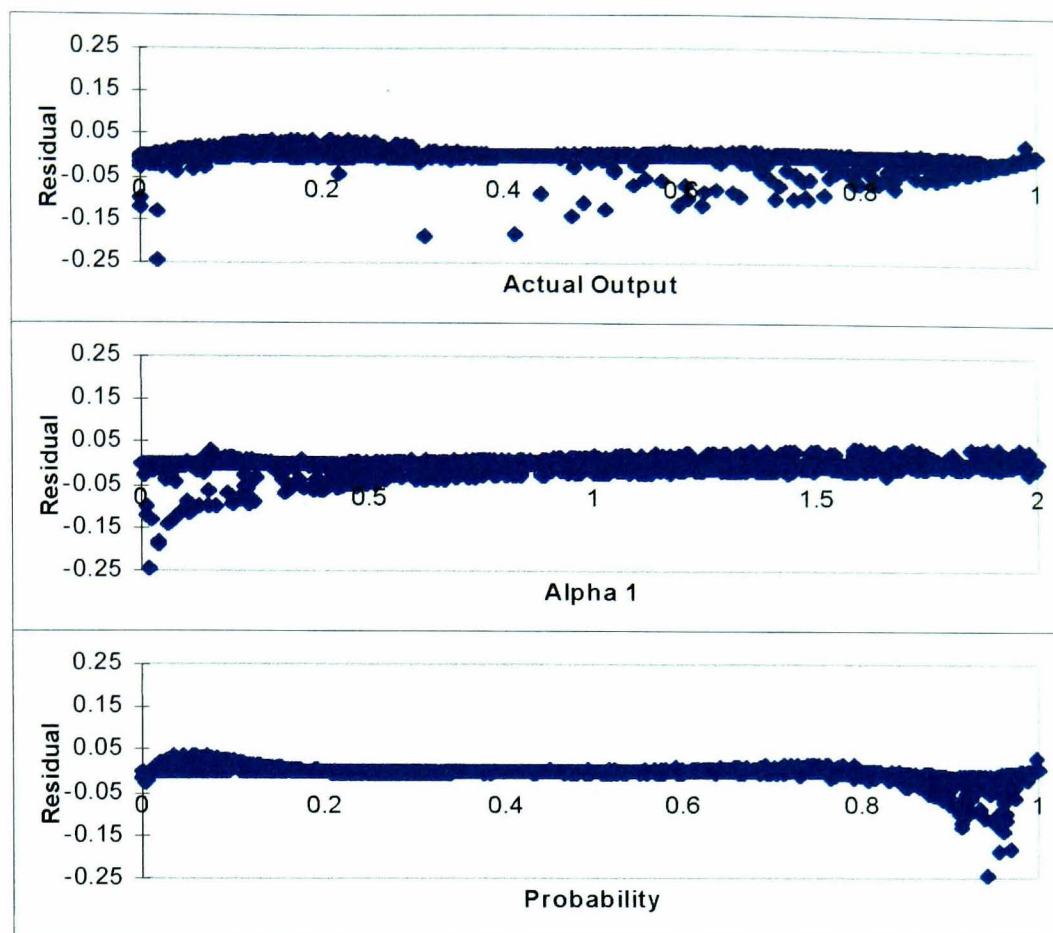


Figure 4.3 : $\text{Beta}(\alpha_1, 1)$: Residual Plots for Small-Structured Data Set (2*8*1)

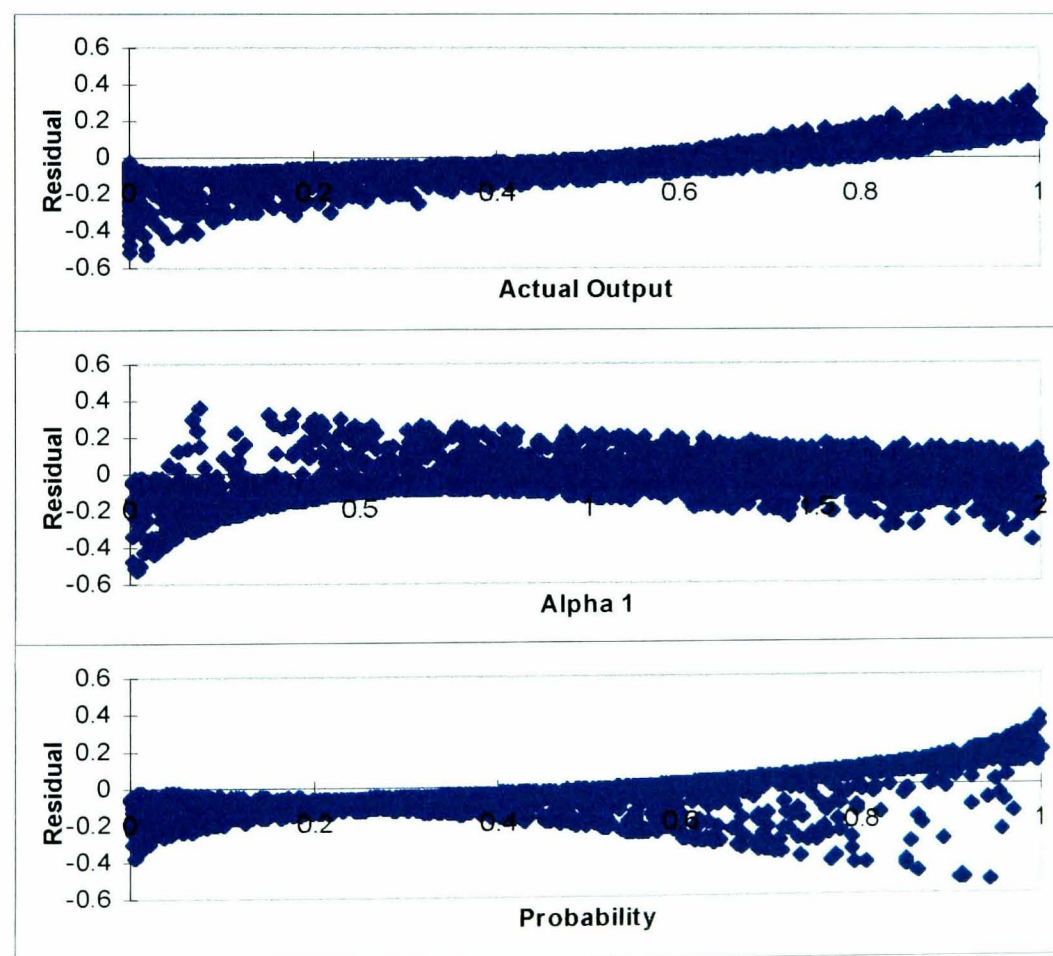


Figure 4.4 : $\text{Beta}(\alpha_1, 1)$: Residual Plots for Small-Random Data Set (2*1*1)

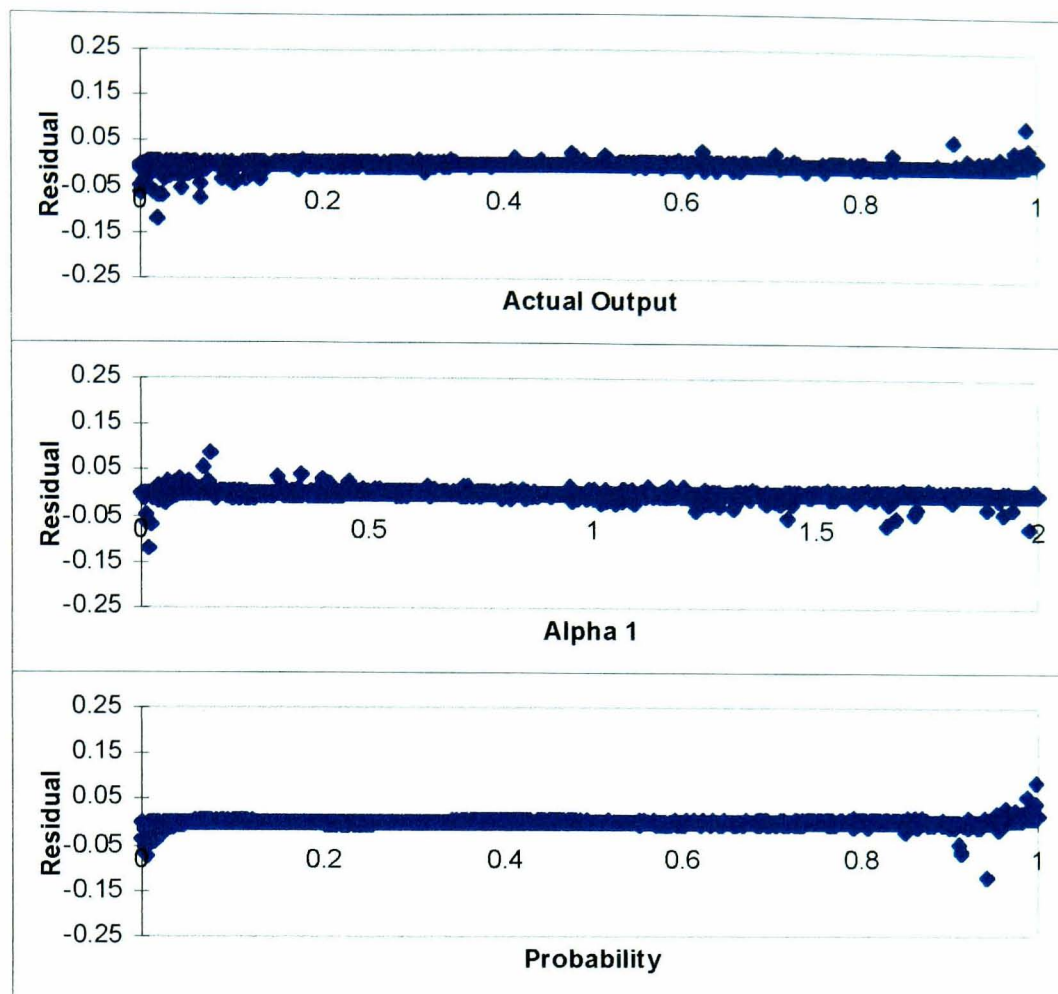


Figure 4.5 : Beta($\alpha_1, 1$): Residual Plots for Large-Structured Data Set (2*8*1(L))

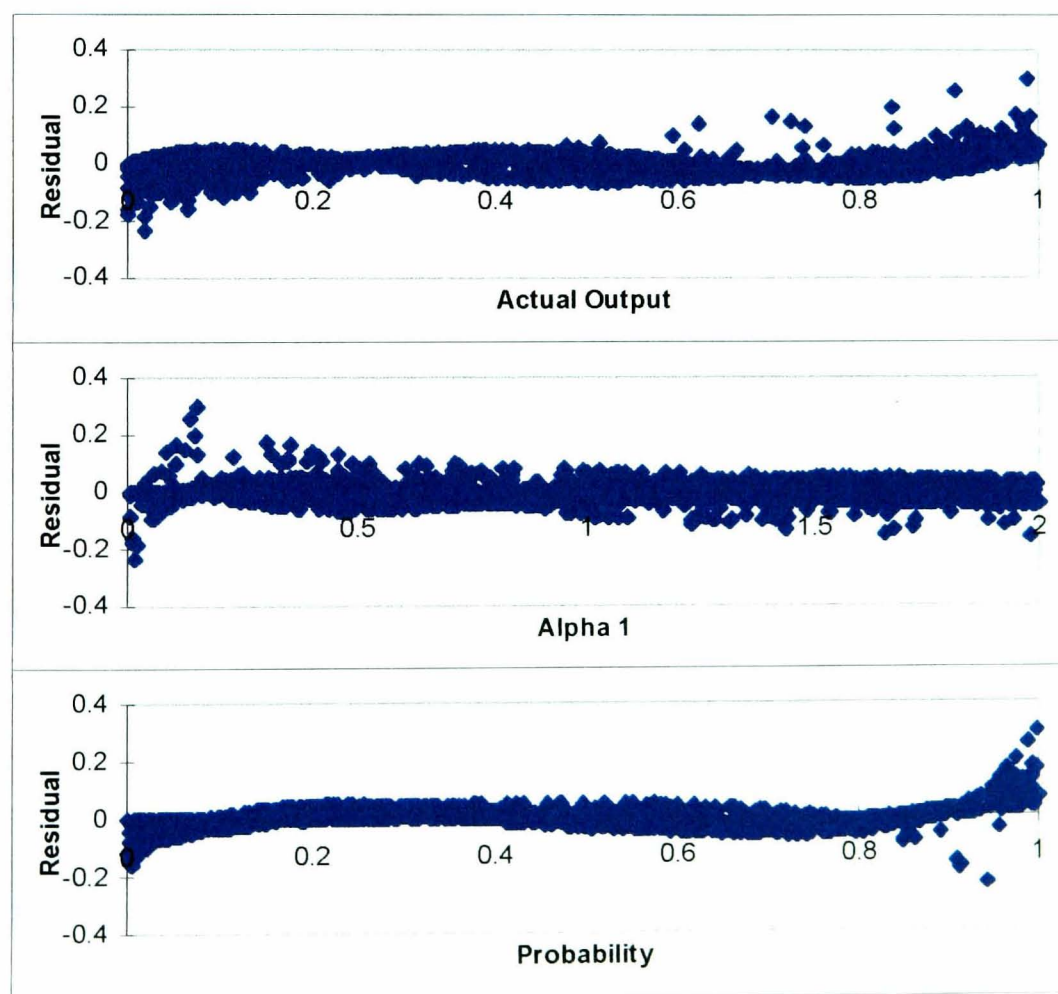


Figure 4.6 : Beta($\alpha_1, 1$): Residual Plots for Large-Random Data Set (2*2*1)

Figure 4.3 shows a reasonably good fit for the small structured data, although the fit is worst for extremely low values of α_1 , particularly where the probability variable is high. This is compared with Figure 4.4 which shows the residuals for the small random data set. This contains a number of fairly large residuals. There is a tendency for the model to overestimate low values for $\text{Beta}(\alpha_1, 1)$ and underestimate high values. There is a marked difference between the structured and random data sets. Since the sample sizes are the same, there is likely to be an effect of using empirical probabilities with a small number of examples in each category.

Figure 4.5 shows the residuals for the large structured data set. This shows a good fit of the model with very few residuals outside ± 0.05 . The residuals are generally worse for low values of α_1 and extreme values for the probability variable. Comparing this against the residuals for the large random data set shown in Figure 4.6, it can be seen that for the empirically generated probabilities, the residuals are much higher. Again low values of α_1 and extreme values for the probability are the main problem areas. There is extreme curvature of the Beta distribution for low values of α , so the modelling task at that point is especially difficult.

It was expected that the results for the data sets with the empirically based probability values would be worse than those with the actual probabilities due to the errors introduced in estimating the probabilities. The results for the small data set are generally poor, but better for the larger data set. Given that the $\text{Beta}(\alpha_1, 1)$ distribution is relatively complex, the results from fitting with the large data set are reasonably good, with a Mean Absolute Deviation of less than 0.03, although the residual plots do show some data regions where the model is not so good.

4.3.4.2 $\text{Beta}(\alpha_1, \alpha_2)$ Distribution

A summary of the MSEs and MADs for the experiments varying both parameters are given in Table 4.6. The results show that in comparison to the structured data sets, the data with empirically generated probabilities show a markedly worse fit to the data. Despite this, the errors for the large data set are still of a reasonable order of magnitude for modelling purposes. However, the errors for the small training set are

fairly large. While the large random data set has the advantage of a finer grid of classes, the errors for the small structured data set are considerably smaller, indicating that good estimates of the probabilities are a key factor in the modelling.

Small - Structured				Small - Random			
	Training	Testing			Training	Testing	
Size	MSE	MSE	MAD	Size	MSE	MSE	MAD
3*4*1	0.001853	0.002312	0.028112	3*4*1	0.020193	0.019312	0.115083
3*8*1	0.000107	0.000215	0.007255	3*8*1	0.016955	0.020126	0.104835
3*12*1	0.000074	0.000202	0.006124	3*12*1	0.017579	0.019324	0.105898
3*16*1	0.000072	0.000183	0.005996	3*8*1(L)	0.016705	0.021419	0.108362
3*12*1(L)	0.000036	0.000161	0.005055				
Large - Structured				Large - Random			
	Training	Testing			Training	Testing	
Size	MSE	MSE	MAD	Size	MSE	MSE	MAD
3*4*1	0.001530	0.002323	0.026827	3*4*1	0.013993	0.003566	0.029744
3*8*1	0.000236	0.000442	0.009067	3*8*1	0.012694	0.002326	0.029432
3*12*1	0.000176	0.000336	0.007295	3*12*1	0.012559	0.002064	0.026446
3*16*1	0.000187	0.000378	0.007715	3*16*1	0.012329	0.002128	0.028393
3*12*1(L)	0.000099	0.000194	0.005705	3*12*1(L)	0.012283	0.002111	0.027772

(L) Refers to network trained for 200,000 / 100,000 iterations

Table 4.6 : Beta(α_1, α_2) : Mean Square Errors & Mean Absolute Deviations of the Neural Network Models

Comparing the results in Table 4.6 with those for the Beta($\alpha_1, 1$) distribution in Table 4.5, shows a reasonable similarity in the general magnitude of errors. The results for the Beta(α_1, α_2) are slightly better for the small-structured and large-random data sets, and worse for the large-structured data sets, while there is little difference for the small-random data set. This suggests that an increase in the dimensionality of the inputs does not have a significant effect on the accuracy, provided a sufficiently large amount of data is available to capture the extra interactions. It is noted that training with more dimensions does seem to require more neurons in the hidden layer.

Figures 4.7 to 4.10 show residual plots for the best network in each category of data set for each of the variables. Note that each plot involves 2000 data points, many of which are collected close to 0 for the residuals.

Figure 4.7 shows the residuals for the small structured data set. Most of the residuals are close to 0, as suggested by the MAD in Table 4.8. The worst results generally occur for low values of α_2 with low probabilities.

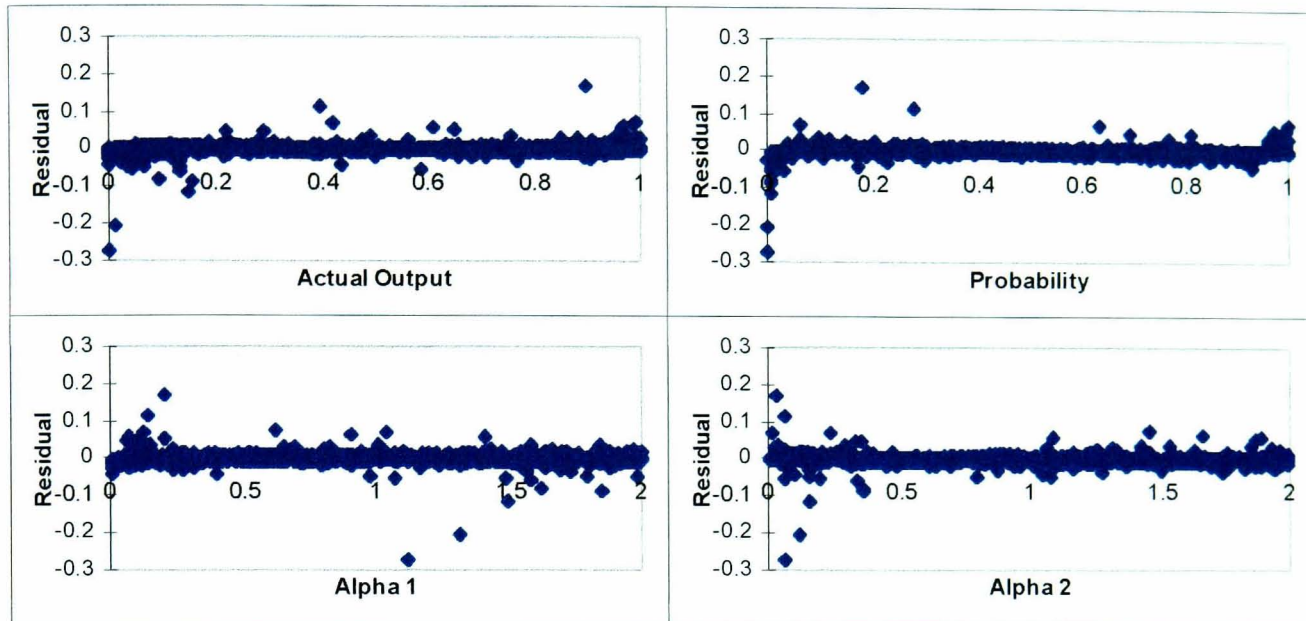


Figure 4.7 : $\text{Beta}(\alpha_1, \alpha_2)$: Residuals for Small-Structured Data Set ($3 \times 12 \times 1(L)$)

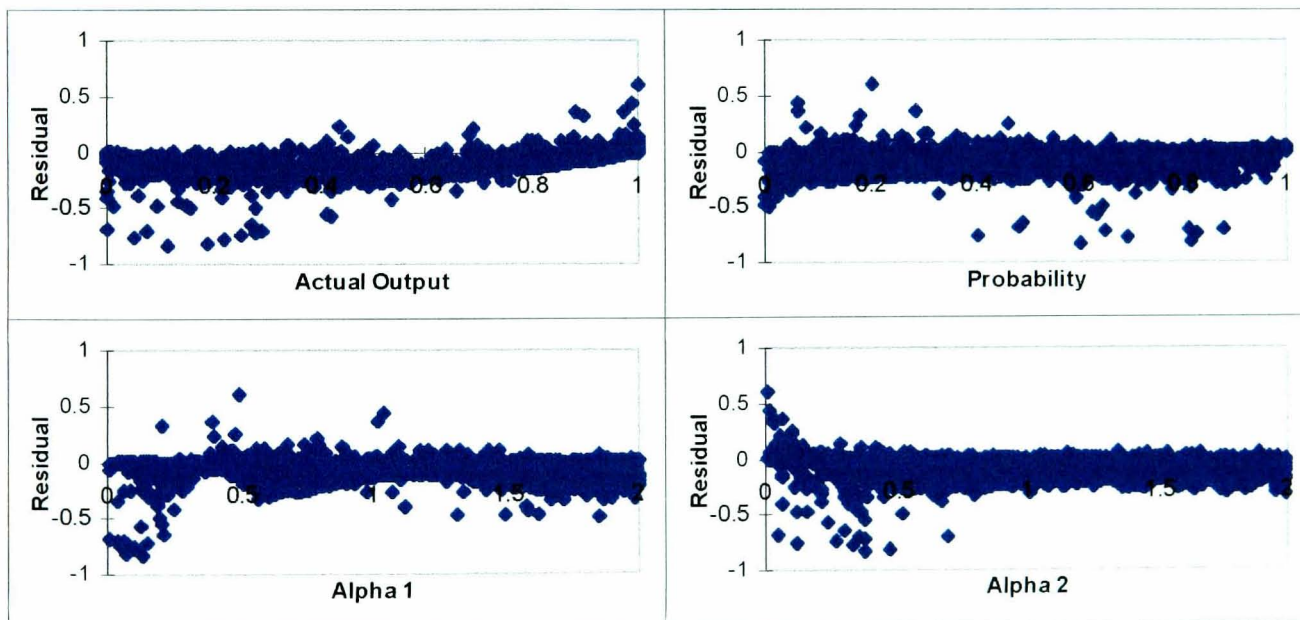


Figure 4.8 : $\text{Beta}(\alpha_1, \alpha_2)$: Residuals for Small-Random Data Set ($3 \times 8 \times 1$)

Figure 4.8 shows the pattern of residuals for the small data set with empirical probabilities. There is a predominance of negative residuals, with some high errors. Particularly high errors occur for some of the low values of α_1 and α_2 . There is no clear pattern with regard to the probability values.

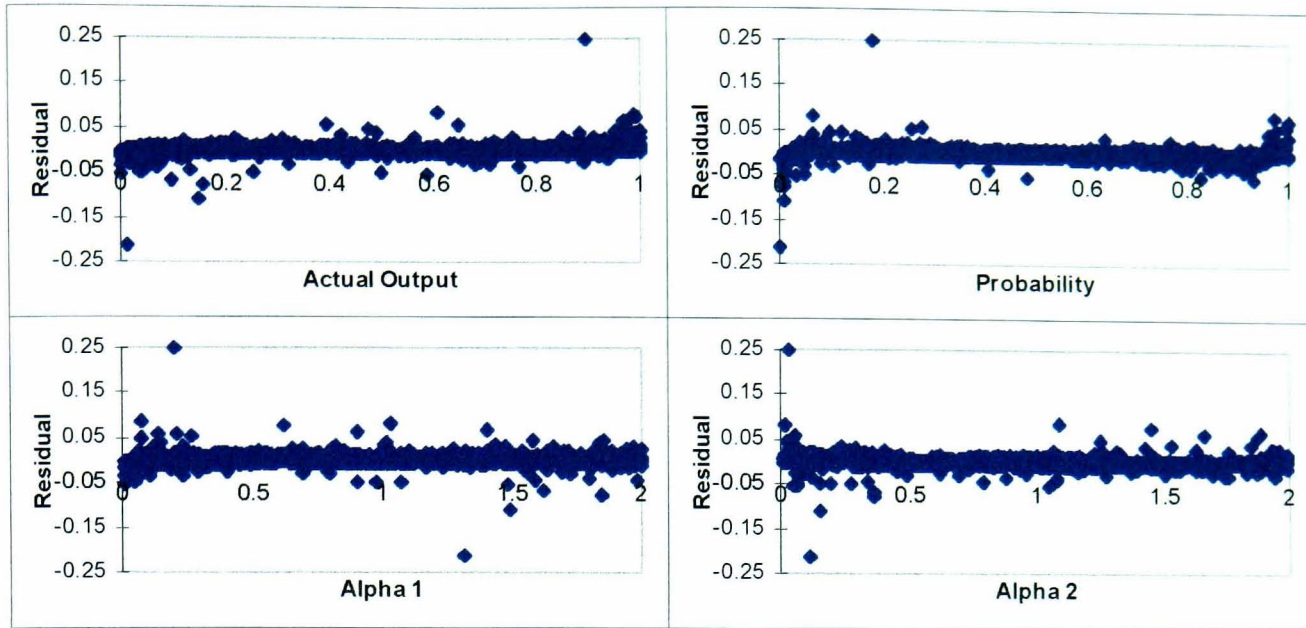


Figure 4.9 : $\text{Beta}(\alpha_1, \alpha_2)$: Residuals for Large-Structured Data Set ($3 \times 12 \times 1(L)$)

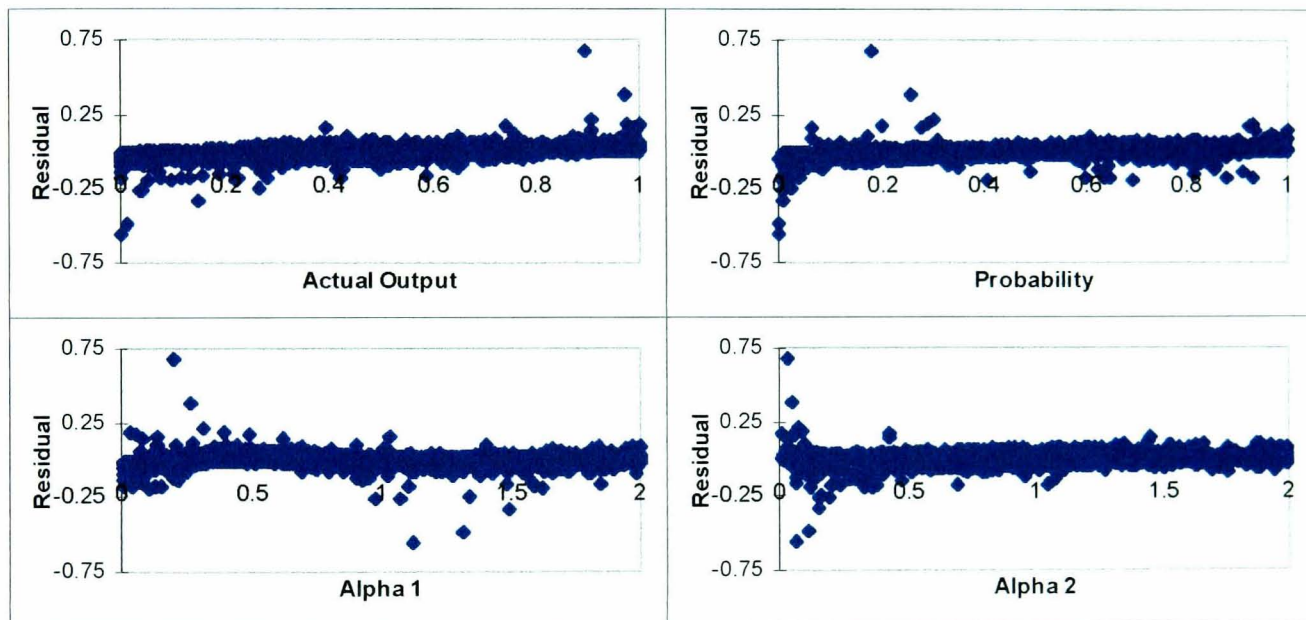


Figure 4.10 : $\text{Beta}(\alpha_1, \alpha_2)$: Residuals for Large-Random Data Set ($3 \times 12 \times 1$)

Figure 4.9 shows the residuals for the large structured data set. The residuals are fairly small, and generally evenly distributed. Some higher residuals do occur for the extreme probability values.

The residuals for the large data set with empirical probabilities are shown in Figure 4.10. These have a fairly even pattern, although some large residuals do occur for small values of α_2 where there are low probability values.

Overall, the residual plots back up the error rates shown in Table 4.6 although they exaggerate the average level of error because the many residuals close to 0 are not shown clearly.

4.3.5 Investigating the Effects of Class Size

The results achieved for the data with empirically derived probabilities were considerably better for the large than for the small data sets. The large data sets had more data classes with smaller ranges, and more data in each class than the small data sets. In comparison, the results for the small data set with known probabilities were considerably better than those for any of the data sets with empirical probabilities. This suggests that the accuracy of the probability estimates was more important than the size of the class intervals for the fitting process.

Having fewer class, with larger ranges means that more data points would be available for generating the empirical probabilities in each class. On the down side, a wider range of raw input data values would be converted to a single class-mid point, and the neural network model would have to do more interpolation between the parameter input values.

A further experiment was done, which investigates the effects on the large Beta(α_1, α_2) data set of using fewer classes to generate the empirical probabilities. The data set contained 4851, but was divided into 11 classes, rather than 21 classes, as before. The class ranges used were the same as those for the small data set, as was shown in Table 4.2. The networks were trained for 30,000 iterations, with the best of those re-trained for 100,000 iterations. The results from the test data are shown in Table 4.7.

Size	Training	Testing	
	MSE	MSE	MAD
3*4*1	0.005332	0.002303	0.025371
3*8*1	0.003956	0.001255	0.019300
3*12*1	0.003976	0.001692	0.020318
3*8*1(L)	0.003785	0.001352	0.018611

(L) Indicates Network Trained for 100,000 Iterations

Table 4.7 : Results for Beta(α_1, α_2) Large Data Set with Large Class Intervals

The results from the experiment show that the accuracy of the model fit to the distribution is considerably better than for the small data set using fewer values to estimate the empirical probabilities. This indicates that the accuracy of the empirical probabilities is of great importance. Furthermore, the results are also better than for the large data set with smaller class intervals. Thus, the increase in the number of examples in each class outweighs the disadvantages of having fewer classes in this case.

The degree of the trade-off between class sizes and accuracy of the estimates of the empirical distributions will depend on the nature of the function to be modelled. A smooth function would be able to have large class sizes, while a complex one with many turning points would need smaller class sizes in order to fit the model accurately.

4.3.6 Overview of Results for Continuous Valued Output Models

Table 4.8 summarises the results from the experiments involving continuous valued outputs, choosing the networks that had the best Mean Absolute Deviation scores on the test data.

Parameters	Data size	Probabilities	MAD	Data size	Probabilities	MAD
Beta($\alpha_1, 1$)	Small	Structured	0.007193	Small	Empirical	0.100149
	Large	Structured	0.002638	Large	Empirical	0.026842
Beta(α_1, α_2)	Small	Structured	0.005055	Small	Empirical	0.104835
	Large	Structured	0.005705	Large	Empirical	0.026446

Table 4.8 : Summary of Results from Continuous Distribution

The results of the experiments for both Beta($\alpha_1, 1$) and Beta(α_1, α_2) give a similar picture. Highly accurate models can be built when the true probability distributions are known, but there is a marked deterioration in the results when the probabilities are determined empirically. Despite this, reasonable results can still be obtained provided that sufficiently large amounts of data are available. As the dimensions of the problem increase (i.e. more decision criteria) a larger the amount of data is required to properly model the interactions between variables. However, provided the larger amount of data is available, there does not seem to be a deterioration in fitting the model.

The larger errors in the model tended to occur for extreme values of the parameters and probabilities. This is partly due to the nature of the Beta distribution, but there is also likely to be an effect of the network training process. Extreme values do not have the weight of other evidence around them that other less extreme values have, and so the network has less information for building a smooth function.

Overall, the results show that it is possible to model parameter-based distributions, but that sufficient amounts of data are required to determine reasonably accurate empirical probabilities. Multiple criteria can be coped with, but more factors require increasingly more data to build the model.

4.4 Representing Classification Outputs

4.4.1 Approaches to Representing Decisions with Classification Outputs

Three alternative approaches were suggested for representing decisions with classification outcomes in Section 3.4.3. These involve 1) an empirically derived probability variable as one of the inputs as used in the continuous output case, 2) pre-processing the data to determine the probability of an outcome in each class and representing these as outputs in the training set, or 3) allowing the network to calculate the class probabilities itself during the training process. The purpose of the investigation is to determine which approach is best for representing decision making.

Each approach will be used on the same discrete distribution to determine the extent of its ability to represent that distribution. Each of the approaches has implications for setting up the training and testing data, and so will be discussed separately below.

4.4.1.1 Training with a Probability Input Variable (INVAR)

Section 4.3.1 discussed an approach for continuous valued variables that involved sorting the input data into classes, sorting the outputs of all data belonging to the same set of classes into ascending order and then allocating empirical probabilities as one of the inputs. A similar approach can be used for the case where the outputs are classification variables. This approach will be referred to as the INVAR approach.

The closest parallel of the classification with the continuous variable approach can be seen if the method of classification is seen as involving a single variable which can take a set of integer values (one for each class). In this way the integer values can be treated exactly as their continuous counterparts in the algorithm given in Section 4.3.1.

Once the empirical probabilities have been determined, the single integer variable can be converted into several binary classification variables, one for each class, as described in Section 3.3.2. Thus, a case with three outcome classes 1, 2, and 3 would be represented by three binary outputs taking values 1 0 0, 0 1 0 and 0 0 1 respectively.

The approach should work best if the classifications follow at least an ordinal scale, that is the classes can be placed in a meaningful order. If this is the case, care should be taken to ensure that the classes are specified in the correct order. This is not essential but the structure of the relationships between the inputs and an ordered set of classifications are likely to be much simpler, and so require a smaller neural network structure to be able to model them.

When the neural network has been trained, a response can be generated by supplying the values for the decision criteria along with a random number. The response of the network may not be a straight forward one where one output has a value of 1, and all the others have a value of 0. Often several outputs will be activated with results of less than 1. In this case the class which is chosen is the one that has the highest output value.

The approach is much like modelling the relationship as an inverse cumulative density function for a discrete distribution. Values for the parameters are supplied, along with a random number, and the outcome class is returned.

4.4.1.2 Training with Class Membership Probabilities as Outputs (OUTDATA)

The discrete nature of the classifications can be made use of to allow a different approach from the case with continuous variables. Rather than using an input variable

to represent the probability, the outputs can be used to represent the probability of a particular decision being made. A random number can then be applied to the result of the network to choose which decision is actually made. This approach is similar to modelling a discrete density function, where the outcome is the probability of occurrence for each class.

The data should be prepared as follows:

1. The data should be prepared into sets of examples, with appropriate numerical values for the decision criteria. For the purpose of the algorithm, it does not matter whether the output classes are represented initially by a single variable taking integer values, or a variable for each class, with each taking a binary value.
2. Sort the data based on the value of the first decision variable. Choose appropriate classes for the first decision variable and allocate each example to a class. The actual values of the decision variable should be replaced by the appropriate class mid-point.
3. *Within* each class, repeat *step 2* for each subsequent decision variable.
4. With the data set sorted into groups with the same values for the decision variables, for each classification variable determine the proportion within the group which have that classification outcome. An output variable is used for each class, taking the values of the calculated proportions within each of the groups.

The stages of data preparation are demonstrated with a hypothetical example. The amount of data used is smaller than would be used in practice, but is sufficient for demonstration purposes.

Stage 1:

The data is organised into examples, and has two inputs and an output. Input 1 is a continuous variable and Input 2 is a classification variable. The output is also a classification variable.

Input 1	Input 2	Output Class
0.23	1	1
0.32	2	2
0.29	1	2
0.16	1	1
0.24	1	2
0.26	2	2
0.17	2	1
0.34	1	2
0.21	2	1
0.29	1	2
0.18	2	2
0.34	1	1
0.26	1	1
0.33	2	1

Stage 2:

The examples are sorted by the value of Input 1. Input 1 is then divided up into two classes with ranges $[0.15, 0.25)$ and $[0.25, 0.35)$, with the examples in each class taking the class mid-point value.

Input 1	Input 2	Output Class
0.16	1	1
0.17	2	1
0.18	2	2
0.21	2	1
0.23	1	1
0.24	1	2
0.26	2	2
0.26	1	1
0.29	1	2
0.29	1	2
0.32	2	2
0.33	2	1
0.34	1	2
0.34	1	1

Input 1	Input 2	Output Class
0.2	1	1
0.2	2	1
0.2	2	2
0.2	2	1
0.2	1	1
0.2	1	2
0.3	2	2
0.3	1	1
0.3	1	2
0.3	1	2
0.3	2	2
0.3	2	1
0.3	1	2
0.3	1	1

Stage 3:

The sorting process is repeated for Input 2. The two class values for Input 2 will be used for dividing into groups. The result is that the data has been divided into four groups.

Input 1	Input 2	Output Class
0.2	1	1
0.2	1	1
0.2	1	2
0.2	2	1
0.2	2	1
0.2	2	2
0.3	1	1
0.3	1	1
0.3	1	2
0.3	1	2
0.3	1	2
0.3	2	1
0.3	2	2
0.3	2	2

Stage 4:

The proportion of examples in each group with a particular output class is calculated. For instance, in the first group, 2/3 examples have the output result of 1, while 1/3 has the result 2, so these form the probabilities. Note that there is now an output variable for each class, and the data examples have been amalgamated so that there is only one data item for each group.

Input 1	Input 2	Output Class 1	Output Class 2
0.2	1	0.667	0.333
0.2	2	0.667	0.333
0.3	1	0.4	0.6
0.3	2	0.333	0.667

This approach results in a reduction in the size of the data set from the raw data. This should result in faster training per iteration. The input variables have to be scaled for use in the network. The outputs should all be in the range (0,1) with the values in each exemplar summing to 1. The neural network in this case will have 2 input nodes, 2 output nodes, with the number of nodes in the hidden layer depending on the degree of non-linearity that needs to be modelled.

When the neural network has been trained, the input values can be applied to produce a response. The response will be the probability of membership for each decision class. It may be necessary to rescale the outputs slightly so that they sum to 1. The probabilities are then cumulated and a random number sampled to determine which class is chosen for the decision. The neural network could be trained directly with data

that has already been cumulated, but it is useful to be able to check the sum of outputs and re-scale if necessary.

4.4.1.3 Allowing the Network to Determine Class Probabilities (OUTNET)

Richard & Lippmann (1992) and Smith (1993) note that the outcomes of neural network classifications can be considered as probabilities, with the probabilities being determined by the neural network during the training process. The network is presented with training data using an output for each class, with only one output activated to 1 (the others being 0) in each example (i.e. for 3 classes this would be 1 0 0, 0 1 0, 0 0 1). Where the data contains conflict between classifications in the training data, several outputs may be activated at less than 1 in the trained network. The network should be able to maintain a probabilistic relationship between the outputs summing to approximately 1.

Two options exist for preparing the data. One is to use the raw input values (suitably scaled) while the other is to classify the input values into classes (as used in all the other approaches discussed in this chapter). The advantage of using the raw values is that maximum accuracy is maintained. Classifying the inputs into groups may however help to improve the structure in the data for determining probabilities.

The application of the trained network would be the same as the case discussed in Section 4.4.1.2, where membership probabilities are determined before training.

4.4.2 Discrete Data Sets

The Binomial distribution was chosen to investigate the ability of the neural network to represent classifications under each of the data representations discussed in Section 4.4.1. The actual distribution used was Binomial($n=4, p$). By fixing n , the number of trials, the situation with a known number of output classes could be modelled (5 classes in this case). The parameter p , the probability of success in each trial, was allowed to vary within the feasible range $[0,1]$. Within the common classical discrete distributions, the Binomial was the best choice because it is the only one that effectively has a fixed number of output classes.

Figure 4.11 shows some example class probabilities from the Binomial distribution. The distribution is reasonably well behaved, with a fairly smooth progression in the classification probabilities as p increases. However, it is still a reasonable test for the ability of the neural network to model discrete decision outcomes. The step function from one class to another at a particular probability value is difficult to pick up near the boundary points.

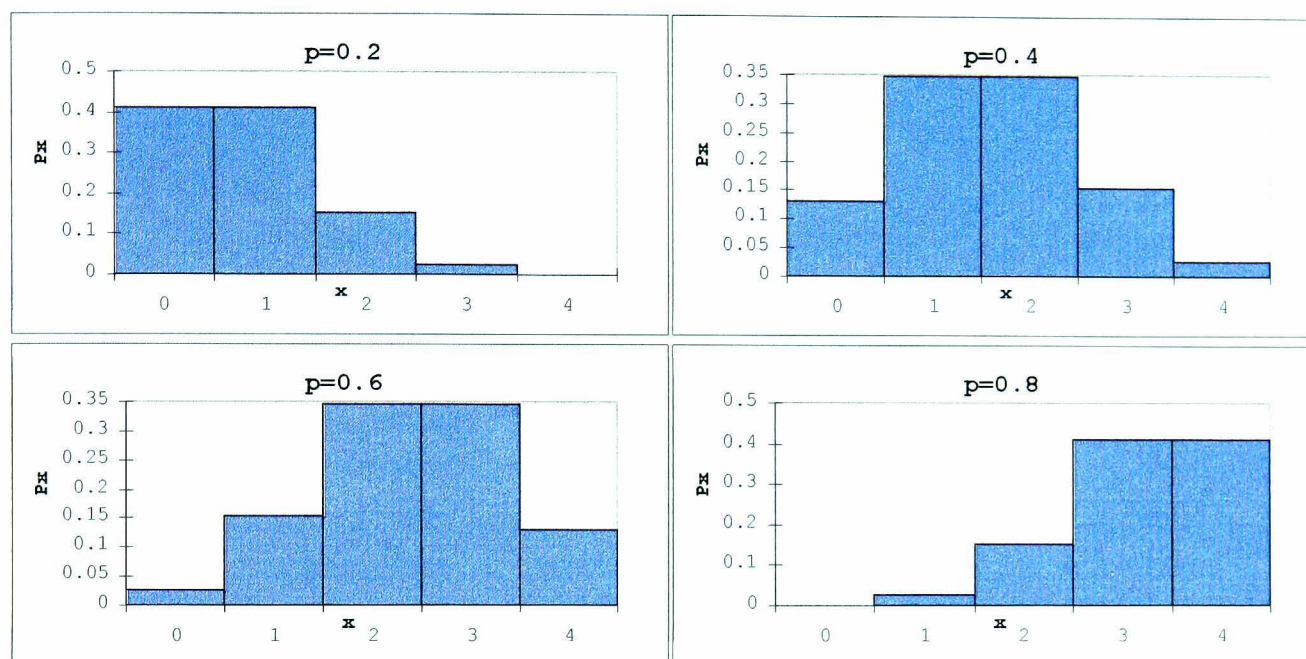


Figure 4.11 : Examples of Binomial(4, p) Distribution

As with the continuous data sets discussed in Section 4.3.2, several data sets were used, each exhibiting different features. Large and small data sets were used. Also there were versions which use the actual probabilities from the Binomial distribution, and versions which derived the probabilities empirically from the data. The aspects of the data sets for each of the approaches to representing the stochastic processes will be discussed separately.

4.4.2.1 Binomial Data Sets using Random Variable as an Input

Data sets with 36 and 121 examples were developed. Table 4.9 shows the values chosen for parameter p and the random variable r for the structured data set with 36 examples. The classification outcome was generated from the inverse cumulative function of the Binomial(4, p) distribution which was developed using an Excel 5.0 spreadsheet.

p	0.05	0.2	0.4	0.6	0.8	0.95
r	0.05	0.2	0.4	0.6	0.8	0.95

Table 4.9 : Input values for Binomial : 36 examples (Small-Structured)

The randomly generated data with sample size 36 had values for p randomly selected from the range (0,1), with the probabilities for generating classifications from the inverse cumulative function of the Binomial selected from the same range. Empirical probabilities for the data set were generated using the approach discussed in Section 4.3.1.1. The class ranges and mid-points chosen for grouping the data based on the value of p are shown in Table 4.10.

Min	0	0.1	0.3	0.5	0.7	0.9
Mid	0.05	0.2	0.4	0.6	0.8	0.95
Max	0.099	0.299	0.499	0.699	0.899	1.0

Table 4.10 : Ranges and Mid-Points for p : 36 Examples (Small-Random)

A similar approach was used for the larger data set with 121 examples. Table 4.11 shows the values chosen for p and r for the structured data set, while Table 4.12 shows the ranges and mid-points for grouping p to empirically generate the probabilities. The data classes allow a finer grid of values for determining the model, as well as having more data available for determining the empirical probabilities.

p	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
r	0.05	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.95

Table 4.11 : Input values for Binomial : 121 examples (Large-Structured)

Min	0	0.05	0.15	0.25	0.35	0.45	0.55	0.65	0.75	0.85	0.95
Mid	0.025	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.975
Max	0.049	0.149	0.249	0.349	0.449	0.549	0.649	0.749	0.849	0.949	1.0

Table 4.12 : Ranges and Mid-Points for p : 121 Examples (Large-Random)

4.4.2.2 Binomial Data Sets with Class Membership Probabilities as Outputs

As described in Section 4.4.1.2, transforming the data to represent the class membership probabilities in the output variables results in a reduction in the size of the

data set. The large and small data sets covering the same range of values as the other approaches have 6 and 11 data points respectively.

The small structured data set used the same values for p as those given in Table 4.9. The values for the output variables were generated from the probability function of the Binomial($n=4,p$) distribution. For the random data set, the same 66 data points as generated in Section 4.4.2.1 were used. The transformation process described in Section 4.4.1.2 used the same groupings as shown in Table 4.10 for the 66 data points and resulted in the data set being reduced to 6 data points.

The large data sets were generated in the same manner as the small ones. For the structured data set, the values of p are as shown in Table 4.11. For the large random data set, the groupings are as shown in Table 4.12. In generating the empirical class membership probabilities, the transformation process involved reducing the data from 121 examples to 11 examples.

4.4.2.3 Binomial Data with Network Attributing Class Membership Probabilities

The data sets used for testing the ability of the neural network to attribute class membership probabilities are the same as those described in Section 4.4.2.1, with the adaptation that the input representing the random variable is removed.

The large and small data sets have 66 and 121 examples respectively. For the randomly generated data sets, there are actually two versions. One version uses the raw values for p , while the other groups the values of p using the ranges and mid-points given in Table 4.12.

4.4.3 Neural Network Training and Testing

Initial investigation of the data sets showed that fewer hidden neurons were required than in the experiments with the continuous valued outputs. For each data set, networks were trained using 2 and 4 neurons, with investigations going on to look at either 1 or 6 neurons if the results from the run indicated that the larger or smaller network might be more accurate.

Networks were allowed to run for 50,000 iterations before being stopped. After this point, the network would be stopped at the point where the total sum of square errors for the training set was at its minimum value for the training period. It was noticed that the mean square errors generally settled quite quickly during the training process, with little improvement being made after that point. It was therefore decided that longer runs of the network were not required.

All of the trained networks were tested with the effectively the same testing data values, although format needed to be changed to suit the method of representing the data that was used. The testing data consisted of 2000 data points with p randomly chosen from the range $(0,1)$ and the random variable chosen from the same range. The true classifications were generated from the inverse cumulative function of the Binomial($n=4,p$) distribution so that the network classifications could be compared with the ones generated by the trained neural networks.

It was noticed during the preparation of the data that the empirical probabilities for the small data sets were a poor representation of the actual probabilities. This was particularly highlighted by the data set which uses the class membership probabilities as outputs.

In the case of the networks that use the random variable as an input to the network (as described in Section 4.4.1.1), the resulting classification can be compared directly to the data. For the cases where the class membership probabilities are represented in the outputs (as described in sections 4.4.1.2 and 4.4.1.3), only the value of p is applied to the network, and the response is the output of the probabilities of class memberships. The random variable that was used for generating the test data was then applied to the network output probabilities to generate the model classification which was compared with the actual classification.

The key measure for success in evaluating the effectiveness of the approaches is the percentage of correctly classified data in the test set, rather than the MSE, so it is these that will be considered in the analysis.

4.4.4 Results of Experimentation

The results for each of the approaches will initially be analysed separately before going on to compare them with each other. The MSE for the training data and the percentage correctly classified for the test data are shown. Due to differences in the way that the training sets were prepared, the MSE can not be compared like with like across the approaches. However, the MSE can be compared within a particular approach to measure the degree of fit to the training data for each network size and so identify evidence of overfitting.

4.4.4.1 Results using Random Variable as an Input

Table 4.13 shows the results for a variety of network sizes for each of the data sets. The network which produced the best classification on the testing data for each of the data sets is highlighted.

Small Structured			Small Random		
	Training MSE	Testing % Correct		Training MSE	Testing % Correct
2*2*5	0.063027	83.80	2*1*5	0.614611	28.25
2*4*5	0.000000*	84.45	2*2*5	0.384722	29.00
2*6*5	0.000000*	84.55	2*4*5	0.279722	14.20
Large Structured			Large Random		
	Training MSE	Testing % Correct		Training MSE	Testing % Correct
2*2*5	0.215124	88.45	2*1*5	0.508264	75.45
2*4*5	0.120702	91.30	2*2*5	0.366664	87.25
2*6*5	0.043273	94.89	2*4*5	0.266124	81.95
2*8*5	0.041504	93.35			

Table 4.13 : Results for Binomial with Random Variable as Input

From Table 4.13 it can be seen that for the small data sets, reasonable results can be achieved with accurate probabilities, as shown by the small-structured, but that the results for the data set with empirically determined probabilities are poor. As noted in Section 4.4.3, the empirical probabilities for the small data set were poor when

compared with the actual probabilities. The results for the large data sets show good classification with the structured data, with only a little deterioration for the empirically generated probabilities. The training MSEs suggest that overfitting is not much of a problem for the structured data sets, but care must be taken when using the data sets with empirically generated probabilities.

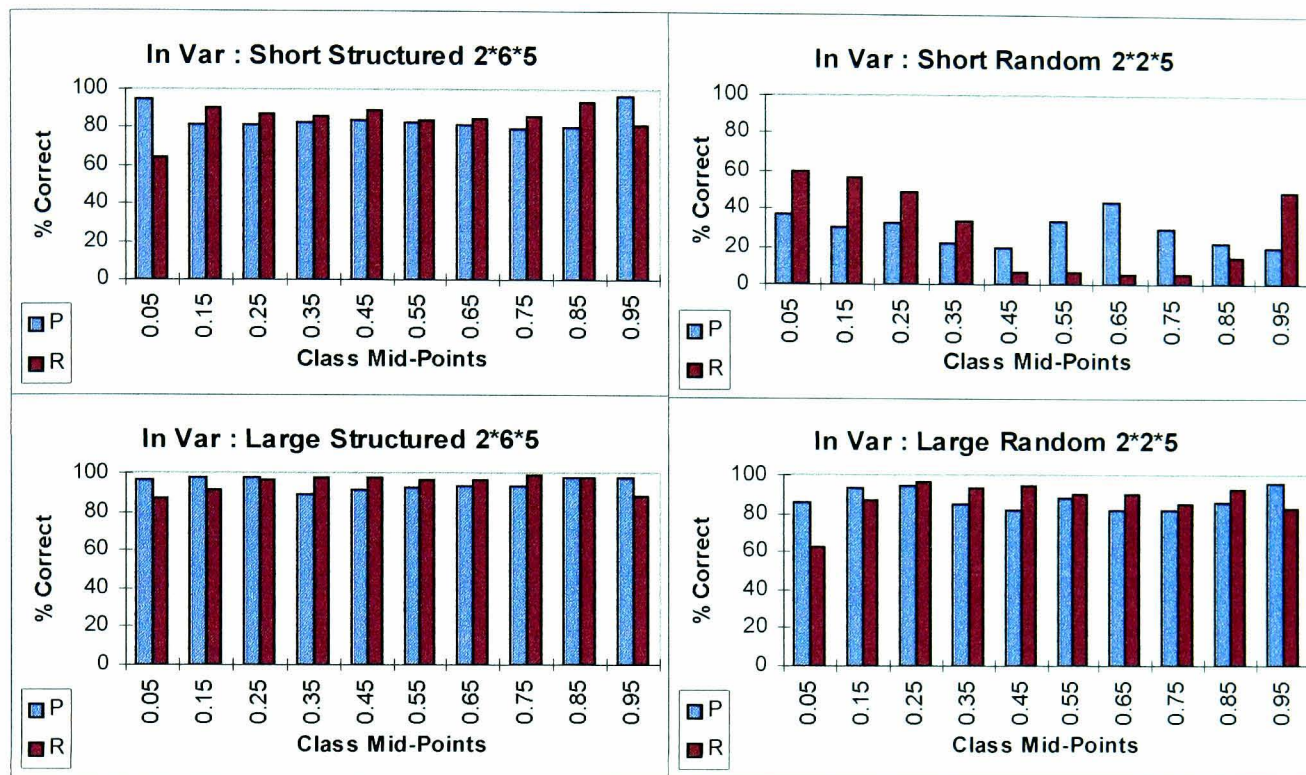


Figure 4.12 : Classifications for Best Networks with Random Variable as Input

Figure 4.12 shows the percentage correctly classified for ranges of probabilities (P) and random variable (R) for the best networks using an input to represent the random variable. Generally, the percentage correctly classified is slightly lower for the extreme values of the random variable R. The exception is the case of the small data set with empirical probabilities, where the opposite effect can be seen. No strong pattern emerges for the probability variable P.

4.4.4.2 Results using Class Membership Probabilities as Outputs

Table 4.14 shows the results for the approach that includes the probabilities of class membership as outputs. The results for the structured data sets show that provided the probabilities are accurate, the class sizes are not so important. However, there is a marked difference when the probabilities are being empirically determined.

Small Structured			Small Random		
	Training MSE	Testing % Correct		Training MSE	Testing % Correct
1*2*5	0.002133	94.80	1*1*5	0.071833	33.35
1*4*5	0.000000	98.85	1*2*5	0.025833	32.30
1*6*5	0.000000	98.40	1*4*5	0.021667	21.95
Large Structured			Large Random		
	Training MSE	Testing % Correct		Training MSE	Testing % Correct
1*2*5	0.001945	95.05	1*1*5	0.107909	74.95
1*4*5	0.000015	99.35	1*2*5	0.046455	80.45
1*6*5	0.000018	99.25	1*4*5	0.010727	73.65

Table 4.14 : Results for Binomial with Membership Probabilities as Outputs

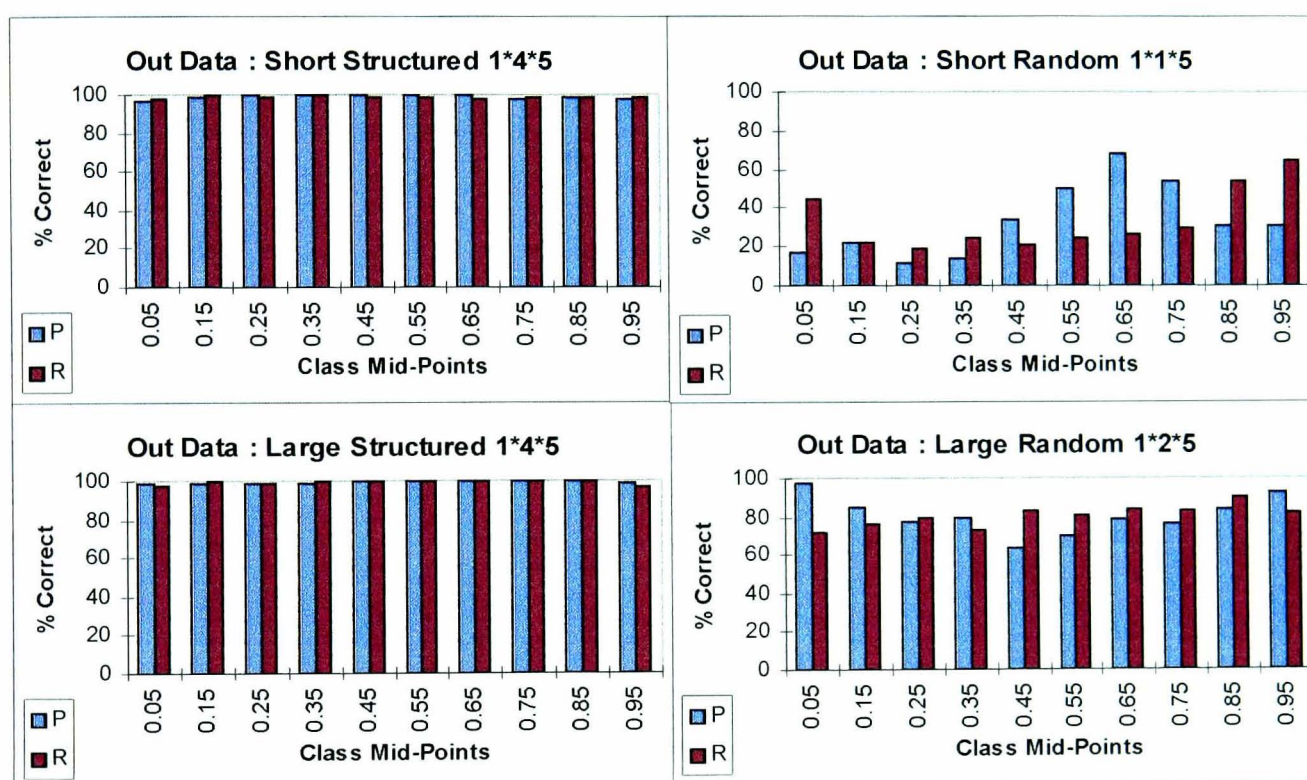


Figure 4.13 : Classifications for Best Networks with Class Probabilities as Outputs

Figure 4.13 shows the percentage of correctly classifications for a range of values of the probability variable (P), and random numbers (R). The graphs show high classifications across the range for the structured data sets. For the small random data set, classification are higher for extreme values of R, but worse for extreme values of P. The pattern is reversed for the large random data set, although the differences are much less pronounced.

4.4.4.3 Results using Network Training to Attribute Probabilities

Table 4.15 shows the results when the network is allowed to determine class probabilities during training. The structured data sets show that when accurate and structured data is available, the size and number of the classes is not such an important factor. However, the results for the data sets with randomly generated values show that the number of data points is important in allowing the network to accurately determine the probabilities.

Small Structured			Small Random (Grp)			Small Random (Ungroup)		
	Training MSE	Testing % Correct		Training MSE	Testing % Correct		Training MSE	Testing % Correct
1*2*5	0.538306	88.55	1*1*5	0.721278	19.05	1*1*5	0.539722	68.00
1*4*5	0.537056	89.15	1*2*5	0.686806	21.85	1*2*5	0.452833	69.00
1*6*5	0.537083	89.90	1*4*5	0.663750	20.90	1*4*5	0.386667	64.70
Large Structured			Large Random (Grp)			Large Random (Ungroup)		
	Training MSE	Testing % Correct		Training MSE	Testing % Correct		Training MSE	Testing % Correct
1*2*5	0.554785	88.85	1*1*5	0.633298	75.75	1*1*5	0.625636	74.50
1*4*5	0.548240	91.25	1*2*5	0.585066	77.00	1*2*5	0.578529	74.25
1*6*5	0.545587	90.15	1*4*5	0.558719	76.60	1*4*5	0.525231	67.70

Table 4.15 : Results for Binomial with Network Attributing Membership Probabilities

The results using grouped data and ungrouped data differ across the data sets. In the case of the small randomly generated data set, the ungrouped approach gave much more accurate results, while the grouped data approach gave slightly better results for the large data set.

Figure 4.14 shows the correct classifications for ranges of values of the probability variable (P), and random numbers (R). The structured data sets show no particular patterns. For the grouped data, the classifications are better for extreme values of P for the large data set, but worse for the small data set. For the ungrouped data, in the case of the small random data set, the patterns for P and R seem to roughly follow the opposite trends, but have a similar trend for the large data set.

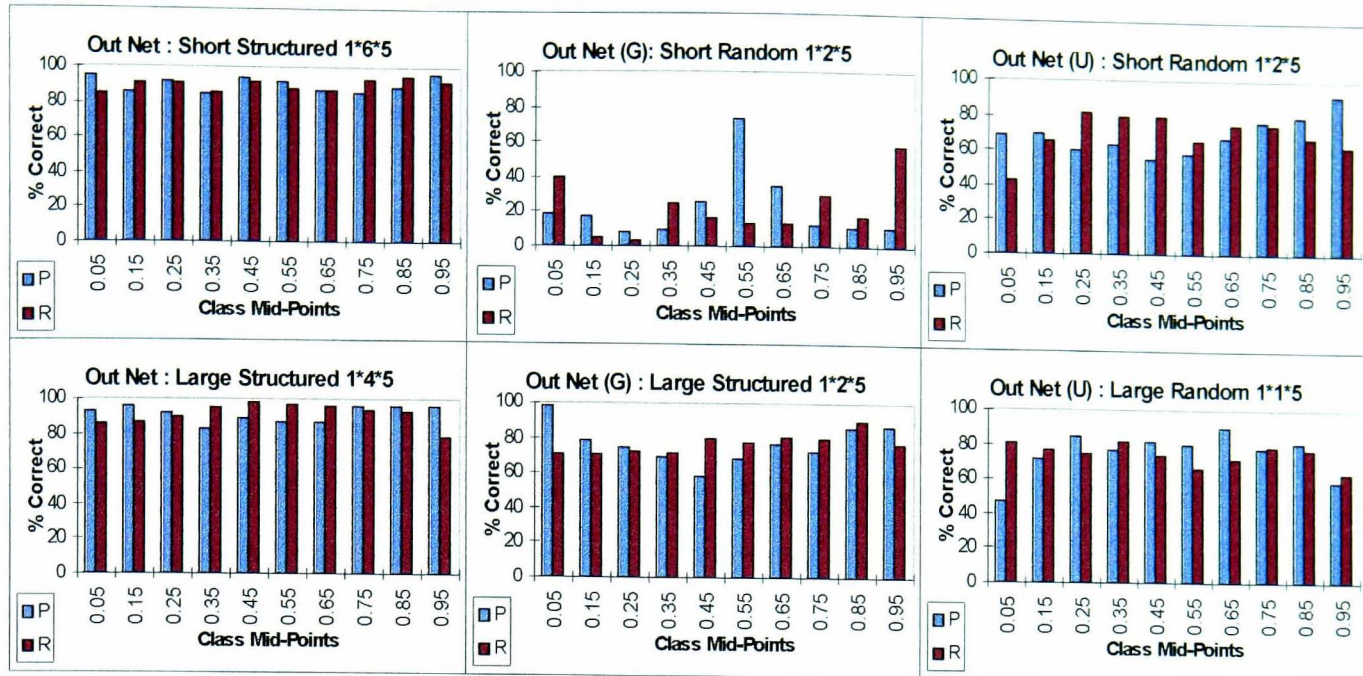


Figure 4.14 : Classifications for Best Networks Determining Class Probabilities

4.4.5 Investigating the Effects of Class Size

It was observed that generally poor results were achieved for the small data sets with empirically generated probabilities. This was considered to be due to inaccurate probability estimates caused by having little data in each group, rather than the large intervals between each group, which was also a feature of the small structured data set.

To investigate the effect of class sizes, a further experiment was carried out that used the same amount of data as the large data set (121 examples), but generated the empirical probabilities using the same class intervals as the small data set (6 classes). The results are shown in Table 4.16.

INVAR		OUTDATA		OUTNET	
Network Size	Testing % Correct	Network Size	Testing % Correct	Network Size	Testing % Correct
2*1*5	73.60	1*2*5	72.85	1*2*5	77.60
2*2*5	84.60	1*4*5	79.85	1*4*5	80.10
2*4*5	78.00	1*6*5	81.30	1*6*5	80.05
		1*8*5	80.80		

Table 4.16 : Results for Large Data Set with Large Class Intervals

The results from the experiment show a considerable improvement over those from the small data set using the same class intervals. In comparison with the previous large data sets with smaller class sizes, the results here are slightly worse for the INVAR

model, and slightly better for the OUTDATA and OUTNET models. This demonstrates that the accuracy of the models is more sensitive to the amount of data available to estimate the empirical probabilities than to the class sizes.

4.4.6 Comparison of Approaches for Modelling Classification Outputs

The best performing networks from each of the approaches are considered together for each of the data sets. The key comparison is in terms of the accuracy of correctly representing the outcome distribution. However, other features such as the work in preparing the data sets and training times will also be considered.

Table 4.17 shows the networks from each of the approaches that produced the best classification rates for each the of the data set types.

Small Structured			Small Random		
Approach	Network Size	Testing % Correct	Approach	Network Size	Testing % Correct
In Var	2*6*5	84.55	In Var	2*2*5	29.00
Out Data	1*4*5	98.85	Out Data	1*1*5	33.35
Out Net	1*6*5	89.90	Out Net (G)	1*2*5	21.85
			Out Net (U)	1*2*5	69.00
Large Structured			Large Random		
Approach	Network Size	Testing % Correct	Approach	Network Size	Testing % Correct
In Var	2*6*5	94.89	In Var	2*2*5	87.25
Out Data	1*4*5	99.35	Out Data	1*2*5	80.45
Out Net	1*4*5	91.25	Out Net (G)	1*2*5	77.00
			Out Net (U)	1*1*5	74.50

In Var : Random Variable as Input

Out Data : Class Probabilities as Outputs in Training Data

Out Net : Class Probabilities Attributed by Network

Table 4.17 : Comparison of Approaches using Binomial Data

The results in Table 4.17 show that in terms of correct classification, different approaches were best depending on the nature of the data set. If the probabilities were known accurately (as in the structured data sets), the approach of representing the probabilities as outputs in the training set (OUTDATA) produced extremely accurate results. Where the probabilities were estimated from a large data set, using a random variable as an input (INVAR) gave more accurate results. In the case of the randomly generated small data set, the approach which allowed the network to determine the

probabilities (OUTNET training the network with the original rather than the grouped data) gave significantly better results than any of the other approaches.

The results suggest that, provided large amounts of data are available, the approach of incorporating the random variable as one of the inputs leads to a better general model than the other two suggested approaches. This also has neatness in that it is effectively the same approach as the one used for the data with continuous valued outputs, and so maintains a consistent approach for both forms of decision. However, if it is felt that probabilities can be calculated with a great deal of accuracy, then it might be better to use the approach of class probabilities as outputs.

The exception is the case where the amount of data is quite limited. Here the approach of allowing the network to determine its own probabilities seems to offer advantages if the data is left in its original form. When data is limited, the loss of information through reducing the input variables to classes appears to be critical.

In terms of preparation and training, the INVAR approach of using a random variable as an input requires the data to be placed into groups and the probabilities calculated. The size of the resulting training data set involves a 1:1 relationship between the raw data and processed data.

The OUTDATA approach of representing class probabilities in the output of the training set requires the data to be prepared, but by amalgamating several raw data points into a single example reduces the size of the training set and so speeds up the training process.

The OUTNET method allowing the network to determine class probabilities involves the least work in preparing the data set, especially if the input data is left ungrouped. Effectively the raw data (suitably scaled) is used. This does mean that the training set size is the same as that of the raw data.

Therefore, while the approach of including the random variable as an input has the advantage of accuracy, it has disadvantages in terms of data preparation and training times compared to the other two approaches.

The patterns of classification rates shown in Figures 4.12 to 4.14 show that except for the cases where there are very low overall classification rates, there is a reasonably even pattern of correct classification. Some minor differences can be seen between the extreme values of the variables and the middle values although there is no consistent pattern to this.

4.5 Independent Validation Data

The experiments described use variation in the size of the hidden layer of the networks to control overfitting, rather than an approach of early stopping through the use of independent validation data. The independent validation approach has been described in Section 3.3.4, and is considered an effective way of preventing overfitting in networks while also being efficient in terms of network training time. The reasons why the approach has not been used are discussed below.

In these experiments, independent validation could have been effectively used for most of the approaches when fitting the structured data sets. A validation data set could have been randomly generated from the distributions and used to determine when training could best have been stopped. The exception is the case of the classification outputs when the network is allowed to determine its own classification probabilities. Here the validation data would have to contain appropriate numbers of examples of the decision outcomes for particular values of inputs so as to be representative of the probability distribution. This would be difficult to generate randomly, and would require either a large validation data set (similar in size to the training set), or specially selected examples from the Binomial distribution.

Despite the possibilities for most of the approaches with the structured data sets, it is the data sets with empirically generated probabilities that more closely reflect the use of the approaches in reality and so are of prime concern. In this case, the validation

data would have to go through the same preparation as the training data. Each of the different approaches has problems with the use of the independent validation data.

For the INVAR approach of using an input to represent the random variable (continuous and classification output cases), the selection of certain data points from training data could not be said to be truly independent, since the value of the random variable would be very much tied in with the probability generation process for the data left in the training set. Removing whole groups of data with the same input values for the other variables would leave large gaps in the training data.

For the OUTDATA approach which represent class membership probabilities as outputs in the training data, the data preparation approach results in a large reduction in the number of examples in the data set. In the experiments, there were too few to be able to create a representative validation data set.

The problems for the OUTNET approach where the class memberships are determined by the network during training have already been discussed for the structured data sets, and hold also for the ones with empirical probabilities.

The limitations and difficulties of using independent validation may be overcome if a large amount of data is available for training (more than was used in the experiments). Provided that sufficient data was available to have many input data groups (and therefore smaller class sizes), the loss of a number of groups to a validation data set would not be significant to the training data. However, in most practical cases the amount of data available is limited. Therefore, the issue of independent validation is not considered further in the thesis.

4.6 Conclusions

Approaches for representing decisions involving both continuous valued outputs and classification outputs were tested using artificial data.

For the continuous valued output case a good fit was achieved to data with the true probability values known. However, in practice this is unlikely to be the case and instead the probabilities will have to be determined empirically. The results from the artificial data show that this can be done to a reasonable level of accuracy provided that sufficient data is available to determine good estimates of the probabilities. The small data set had on average 5 data points per combination of classes, but did not lead to a satisfactory model. The large data set with 10 data points per combination of classes, and more classes led to a more satisfactory model. While the average error of this model was low (mean absolute deviation of 0.026 for both the one and two parameter cases), some high residual values could still be observed.

Comparing the cases with one and two parameters, adding extra dimensions to the input criteria does not lead to a deterioration in the model provided sufficient extra data is available to properly represent the increased number of combinations of factors. Available data could be a limit to the complexity of model which can be built.

The case of using classification outputs involves the use of the same neural networks as for the continuous outputs, but more options exist for modelling the random variation. The results indicate that there is no one approach which is superior across the range of data sets.

Where the probabilities were known with high accuracy, the OUTDATA approach representing class membership probabilities as outputs achieved extremely high classification rates with both small and large amounts of training data. However, where the probabilities were being derived empirically the approach did not do as well compared to others.

Where a reasonably large amount of data was available for determining probabilities, the INVAR approach using a input variable for the random variation produced the best results. Where data was more limited, the approach of allowing the network to determine its own probabilities from the data was vastly superior to the others, provided the input data was left with its original values and not placed into classes.

This suggests that when data is limited, the loss of detail through grouping input values has a large detrimental effect on the accuracy of the model.

The models were shown to be more sensitive to the accuracy of the estimates for the empirical probabilities than to the size of the class intervals. Thus, when the data is limited, the results suggest that it is better to have larger class intervals than severely limit the number of examples in each class. The degree to which this can be generalised to other data sets is limited by the smoothness of the relationships between the variables. For functions with many turning points, large class intervals will have a detrimental effect because of a lack of homogeneity between examples in the same class. For smoother functions, the level of homogeneity will be greater, and interpolating between class mid-points easier.

The errors and classification rates for the data with empirically generated probabilities are the key ones to consider. For the larger data sets these are good, but not outstanding. However, when considering the difficulty of the task, the approaches do show some promise. The results indicate that the process of deriving the probabilities empirically has a significant effect on the accuracy of the models compared to knowing the probabilities with perfect accuracy. However this will be an issue with any approach for modelling the variability. It should also be considered that while the Binomial distribution is a relatively straight forward function, and the Beta rather more complex, the task of trying to represent these without knowledge of the underlying function would be extremely difficult using a conventional abstraction or rule-based approach.

Overall, the results of the experiments with artificial data are encouraging. However, the models with artificial data can be easily specified. Chapter 5 goes on to look at the implications of using the approaches for practical model of decision making.

Chapter 5

Issues for the Practical Use of Stochastic Neural Network Models

This chapter looks at some issues for using stochastic neural networks in practice which have not been examined in the analysis on artificial data. Section 1 examines the task of representing the decision making situation in a form that can be used by the neural network. Section 2 discusses the use of a hybrid system that combines the neural network approach with a rule-based one.

5.1 Encoding the Situation

5.1.1 Specifying Variables

Using the neural network approach to representing decision making requires that the decision making situation is converted into a numeric representation. Effectively the problem can be split into two parts : 1) identifying and coding the decision criteria, and 2) identifying and coding the decision outcomes.

The decision criteria are the parameters which are thought to influence the behaviour of the decision makers in the system. These can only be factors that are observable and can be represented in the simulation. Criteria that take on a small number of discrete states (e.g. open/closed) can be represented by classification variables. Criteria that take on a greater number of discrete states or integer values (e.g. queue lengths), or continuous values that can be measured (e.g. car velocity) can be represented by real valued variables.

Hoptroff (1993) claims that one of the advantages of neural networks is that speculative variables (those that might or might not have significant explanatory power) can be used without diminishing the effectiveness of the network. While this might be the case, the side effect is that more data is required for determining the

empirical probabilities. Therefore careful investigation of the possible decision criteria is required by the analyst so as to make best use of the data available.

The decision outcomes are generally easier to identify since these are the focus of the model. However, it is still necessary to determine specifically what format the decision takes and how it affects the action of the entity. Decisions that result in controlling some process (e.g. car speed) involving either continuous or integer values can be represented by real valued outputs and so fall into the class of problems discussed in Section 4.3. Decisions that result in alternative behaviours can be said to form a finite class of mutually exclusive behaviours. Even where several behaviours can occur at once, the set of possible combinations can be said to be mutually exclusive. These problems fall into the classification output class of problems discussed in Section 4.4.

There are issues of how encompassing the model of decision making process is, i.e. how many sub-decisions are involved in coming to the overall decision, and whether or not those sub-decisions should be explicitly modelled. This is considered further in Section 5.1.2.

An issue for both criteria inputs and decisions outputs is one of scaling for use in the network. In most cases scaling to the range (0,1) is reasonable for both inputs and outputs. The key decision is what to scale the data from. Each variable can be considered independent as far as scaling is concerned, unless there is some structural advantage to keeping a 1:1 relationship between a pair of variables. The training data provide clues as to what range to scale from but it should be remembered that these ranges will need to be used with the trained network within the simulation model. Therefore it is better to consider the range of *possible* values that the variable can take. In some cases, such as queue lengths, there is no theoretical upper bound, but in these cases a reasonable cut-off point must be chosen.

While a number of issues in encoding the situation have been considered, a key factor which can affect the nature and scope of the model is the availability of suitable data. The ability to collect large amounts of data or particular data items can be limited by

the time available to collect it, cost of collection and the feasibility of measuring variables. The impacts of having limited data are discussed in sections 5.1.2 and 5.1.3.

5.1.2 Specific vs General Specifications

The final decision outcome might be highly specific, but might involve going through some sub-decisions which are less specific. Thus, with the proper formulation of the problem, it might be possible to produce a more general model. The main advantage of this is that the general model is likely to require less data to model the relationships, since the data that is available is more widely applicable.

The concept of generalising decision making models was initially explored by thinking about decisions that might be made for an existing simulation model. This is a visual-interactive simulation model of a petrol station (based on a real situation) that was built for the Mentor computer-aided learning module on simulation (Mentor, 1993). The petrol station sells three different types of fuel, two of which are available from each pump, and offers no other services.

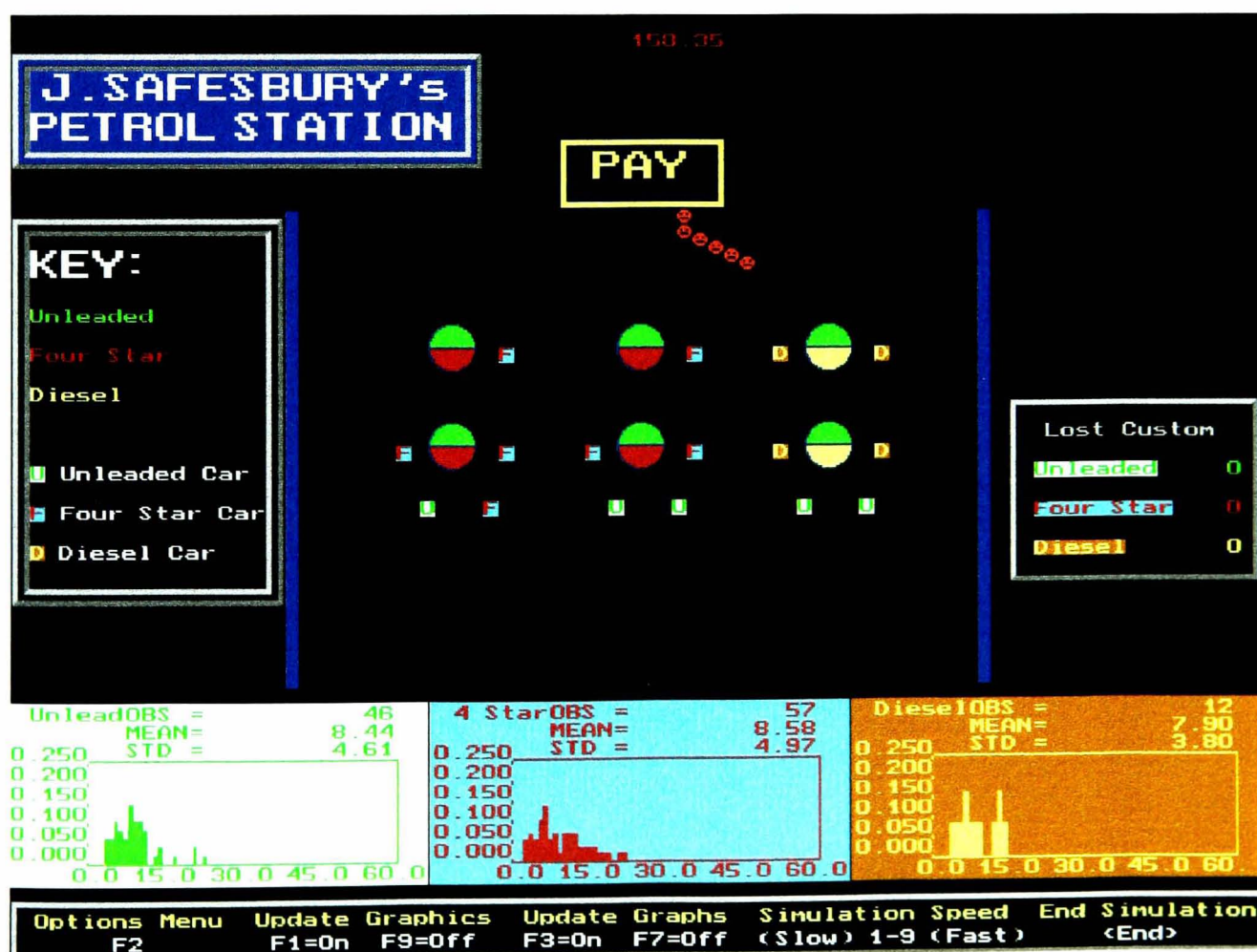


Figure 5.1 : Screen Display of the Petrol Station Model

Figure 5.1 shows the screen layout of the petrol station model. The petrol station has six petrol pumps, each of which can be accessed from both sides. The pumps are arranged in three rows each having two pumps. The dimensions of the petrol station mean that a customer using one of the pumps in a row can impede the use of the other pump, either by preventing access to the pump, or preventing the other vehicle from leaving the petrol station. A car can be seen to be prevented from using a vacant pump in the left-most pair of pumps.

The original petrol station simulation has a number of rules for deciding which queues cars join, based on the petrol available, which side of the car the petrol filler cap is, and the size of queues. It also has simple rules for balking that take into account queue sizes.

The decision to be modelled might be the queue that a car would join or whether it balks. Six possible queues exist, with balking as the other option so that these could be represented as seven classification output variables. The decision criteria could involve : the sizes of the six queues, the state of the use of each pump (empty, in use, customer paying, unavailable), the number of pay counters open (one or two), the size of the pay queues, the type of petrol required, the types of petrol available, the side of the petrol filler cap. When each of the pumps and pay counters is treated separately this amounts to 23 input variables, about each of which data needs to be collected, and each of which is a separate factor for determining empirical probabilities.

A situation might exist where only Pump 1 is free (with the right sort of fuel), and there is no queue. The data might suggest that in those conditions the cars always go to Pump 1. However, the model formed is context specific to Pump 1, and does not generalise to the same situation for Pump 5, unless separate data exist that suggests this. With 23 decision criteria, some with a number of levels, taking account of the variety of situations requires a substantial amount of data.

Several opportunities exist to generalise and simplify the model. Some variables can be combined. For instance, the petrol required variable and the petrol available variables could be combined to determine which pumps are available to a particular car without being specific to its fuel type. The choice of staying at the petrol station or balking could be treated as a sub-decision and modelled separately with far fewer variables, such as taking account of the length of the shortest queue rather than all of the queues. Some of the obvious situations (such as empty queues) could be handled by rules, with the other situations where patterns and variability are important being handled by neural network models.

In some cases, the abstraction of the problem to make it more general may result in a loss of representational accuracy, but this needs to be offset against the reduced data requirements.

5.1.3 Observed and Possible Behaviour

The neural network, being data driven, relies on observations to model behaviour. This works for common occurrences since sufficient numbers of observations of these should be available. However, with more unusual occurrences there is a lower chance of these being observed. This is a problem for any fitting of empirical distributions, and holds for the data driven modelling of behaviour.

The analyst needs to use knowledge of the real situation (theirs or other peoples) to determine what occurrences are represented in the data, and what other occurrences could possibly happen but are not represented in the data.

It is common that unusual occurrences happen under extreme conditions that do not normally occur. This has two advantages: firstly that the occurrences will rarely occur in the simulation, and so small inaccuracies in the model are less likely to have a major impact on the simulation results than errors for conditions that regularly occur; and secondly that the behaviour in extreme cases will often be quite predictable and have low variation. The predictability of the behaviour is likely for several reasons. One is that out of the ordinary events which will not have been experienced by the decision

makers are likely to require more conscious consideration of the decision, and so can be more readily captured by rules. Secondly, the extreme circumstances might constrain the number of options that the decision maker has.

Two options exist for representing the unusual occurrences. One is to add artificial data to the network training set so that the behaviour can be learned. This could be particularly useful for behaviour with a very low probability of occurring. The other option is to add rules that supersede the network if extreme conditions occur. The mechanisms for implementing such a system are discussed in Section 5.2.

5.2 Hybrid Neural Network / Knowledge-Based System

5.2.1 Reasoning Behind a Hybrid Approach

As discussed in Section 5.1, there are situations where a more complex arrangement than a single neural network is required. Several neural networks may be used which model sub-goals towards an overall decision, or where rules may be used to deal with obvious decisions or unusual events. This requires a more complex mechanism and some overall control of sub-components of the decision making model.

An approach for organising and controlling the decision making system can borrow some ideas from blackboard systems. Englemore *et al.* (1988) describe a blackboard model as “a relatively complex problem-solving model prescribing the organisation of knowledge and data and the problem solving behaviour within the overall organisation”. The blackboard system contains a number of independent stores of knowledge which can be represented in different ways. The data describing the problem, and any partial solutions to the problem are stored in a central blackboard. A control unit or scheduler controls the sequence in which the separate knowledge sources use available data to work towards a solution. Although most authors limit their view of knowledge modules as being rule-based or procedural based, there is no reason why a neural network cannot be used as one or more of the knowledge representation modules.

Thus a hybrid system, based on neural network and knowledge-based system modules being co-ordinated by a control mechanism, is suggested. This would allow a complex decision to be split into simpler sub-problems which could be modelled using the most appropriate methods. Also if the problem is split sensibly, it can help to generalise the decision more than by representing the whole problem with a single neural network.

5.2.2 Organisation of the Hybrid Model

Figure 5.2 shows an overview of the neural network / knowledge-based hybrid. The diagram shows only a single rule-base and neural network module, but several of these can be used.

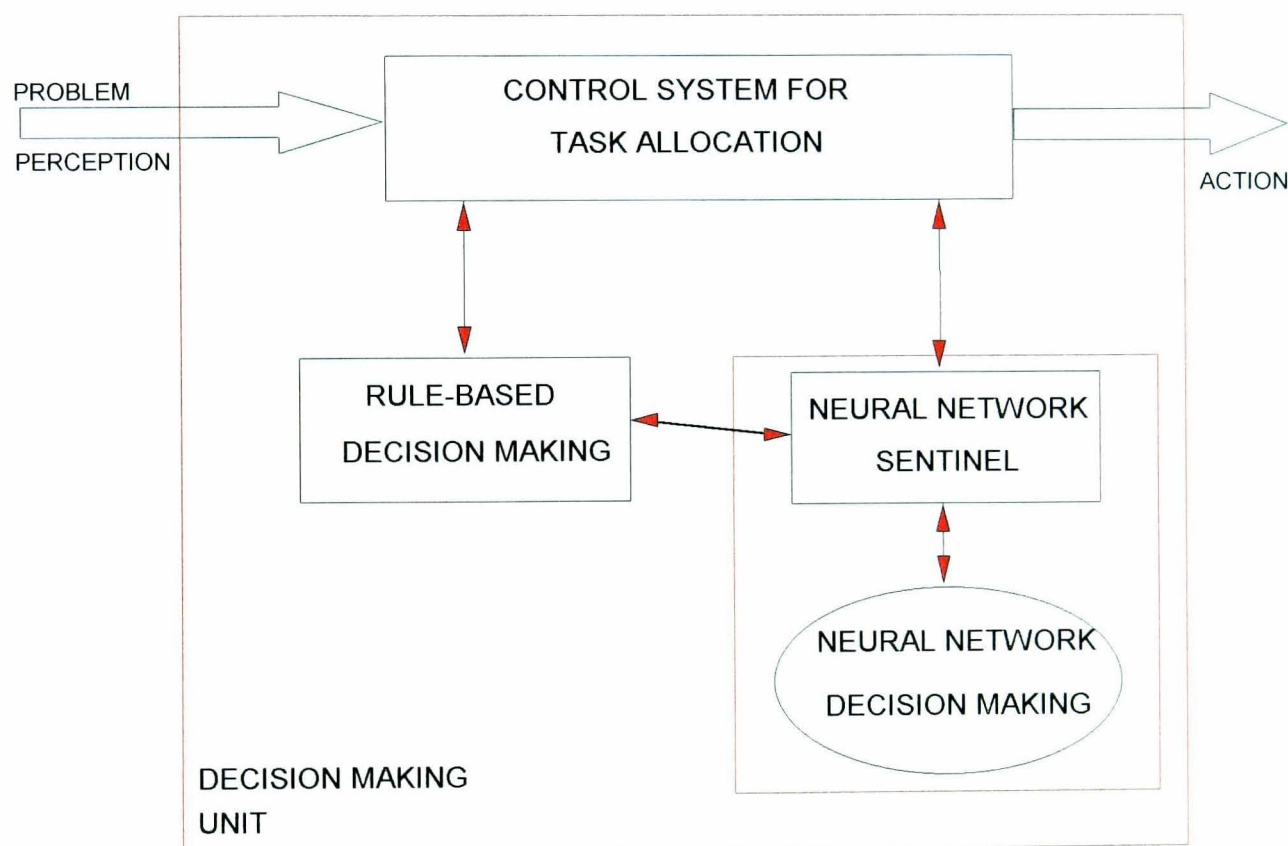


Figure 5.2: Neural Network / Knowledge-Based Hybrid

The relevant data for the decision making is requested by the relevant parts of the hybrid system. This is processed and the resulting decision output is then passed back into the simulation to be acted upon.

The task allocation unit holds the data for decision making and any partial solutions, and controls which knowledge modules will act on the data. This might require examining the data to determine which approach is most appropriate. The neural network module involves a sentinel and the network itself. The sentinel has the job of preparing the decision data so that it is in the right format for use by the network. There may also be a task of validating the output of the network to ensure that the results are within acceptable bounds. The rule-based module allows the use of logical rules of behaviour where these are more appropriate than using the neural network. The main components of the hybrid system are discussed in more detail below.

5.2.3 The Task Allocation Unit

A task allocation unit co-ordinates the solving of the sub-problems. Each sub-problem is dealt with by a different rule-based or neural network component, so that the task allocation unit needs to pass the relevant pieces of information to each component. Since the results from one sub-problem may be needed for consideration of another, the task allocation unit is also responsible for sequencing.

When the results for all the sub-problems have been found, the task allocation unit will collect the final decision and pass this onto the simulation for action to be taken. The decision may be treated as instantaneous, or may have some time duration associated with it.

Since the decision making module will tend to be quite specific to a particular decision that needs to be made, the task allocation unit will usually be fairly straight forward program code. Certainly it would be much simpler than a controller in a conventional blackboard system since that has to cope with a much wider range of queries.

5.2.4 Neural Network Modules

The neural network modules contain trained stochastic neural networks for representing the decision making behaviour. The format of decision criteria and decision outputs may need to be changed so that they match the formats that were used in training the networks (including the scaling range). A key job of the neural

network sentinel is to handle the process of ensuring that data and results are in the correct format. Another possible job for the sentinel is to check the output of the network to ensure that it is within acceptable bounds for the problem context, e.g. for a car driver simulation, that the deceleration of the car does not exceed the maximum braking capacity of the car.

The neural network software, developed in Borland Pascal (described in more detail in Appendix A), allows a version of a network to be used in other Pascal programs. The network does not allow training to be performed, but the dynamics of trained networks can be loaded, and used with new data to produce responses. Several different networks can be used at the same time within a program.

5.2.5 Rule-Based Modules

The rule-based modules contain rules for the decision making behaviour of entities in the system. These rules cover the special cases that are not included in the neural network modules. The special cases may be unusual events, or where behaviour is so predictable that the probabilistic abilities of the neural networks are not required.

On the whole, it is envisaged that the rules employed will be simple and few enough that they can be expressed in the language of the simulation software. If necessary a logic-based language such as Prolog could be used to express the rules, and be interfaced with the simulation software. Flitman (1986) investigated the process of linking Prolog with a procedural language. The use of Microsoft Windows Dynamic Link Libraries, makes the process of linking different application languages together much easier. In this thesis, the simulation system used is DOS based (using Borland Pascal) and so linking this with Prolog is problematic. Therefore, in this work, ordinary procedural code will be used to express the rules.

Chapter 6

Testing the Neural Network / Knowledge-Based Hybrid Model in Practice : The Bank Simulation

This chapter examines the use of stochastic neural networks, and the hybrid decision making model by using them in a simulation application. The application chosen is a simulation of the front office activities within a bank. The exercise will attempt to model the decision making behaviour of the customers. This chapter will look at the modelling process in detail for the chosen application, with a more general discussion of the approach taking place in Chapter 7.

Section 1 outlines the aims of the study, and discusses the approach taken. Section 2 describes the Bank Simulation model. Section 3 discusses the initial pilot study where the decision making behaviour of just one person was modelled. Section 4 describes the approach taken for collecting decision making data from a total of twenty subjects. Section 5 covers the data analysis and model building phases of the study, while Section 6 describes the implementation of the models into the Bank Simulation. Finally, in Section 7 there is a discussion of the observations and lessons learned from conducting the study.

6.1 Choosing the Application

The key aim of the study is to examine the practice of using stochastic neural networks, within the framework of the hybrid neural network / knowledge-based system. Chapter 4 looked at the performance of the stochastic neural network approach on artificial data, while Chapter 5 discussed some of the issues that arise when thinking about the practical application of the suggested approach. However a fuller understanding of the applicability of the approach, its strengths and its weaknesses, can only be achieved by applying it in practice. Even with the use of a practical example, it should be borne in mind that this is a prototype for a single

system, and so careful thought must be given to those lessons learned which are specific to the problem, and those which can be more widely applied.

The key difficulty in setting up a practical example is having access to meaningful data for developing the model. This is particularly true for modelling decision making since there are a number of factors which have to be kept track of, making data collection a complex task. There are organisational issues of having access to a suitable system for testing out the ideas, and technical issues of actually collecting the data. The technical issues are dependent on the nature of the system to be modelled. Electronic tracking of system states would allow data to be collected with relative ease, and such a facility may be available for situations such as modelling the behaviour of telecommunications users. Video technology might be of use for observing and analysing physical behaviour, such as customer queuing or car driver behaviour. Unfortunately this technology is expensive, and would be difficult to justify for an untried method of modelling, but is a possibility for the next stage of development.

A solution to the problem of access to, and the collection of decision making data is to make use of a computer model with which subjects can interact. This means that there is control and access to the system, and a mechanism can be built into the model for the collection of data. While the environment is artificial, the decision making is real and so provides valid data for analysis.

The use of a computer controlled environment may actually provide a solution for data collection in real world applications, provided that it is felt that the simulator will induce behaviour similar to that in the real world. For instance, aircraft or car driving simulators could be used to collect data on the responses of people to various situations. Such an approach is suggested by Williams (1996) who describes the use of a simulation for collecting information on intelligent decisions for ship replenishment at sea. The simulation required input by users, and the the decision making was observed for the purpose of developing an expert system.

The front office activities of the bank were chosen as the application to model. From the decision making perspective, it was behaviour of customers in the bank that was of particular interest since it is these who are likely to exhibit variability in their decisions. A bank was chosen for several reasons, which are explained below.

Firstly, the subjects from whom the decision making data will be collected from are almost certain to have had regular experience of being customers in banks. They are likely to have some implicit (and sometimes explicit) view on how long, or how variable different bank services would usually be. Secondly, in many cases the business that customers have in banks is not urgent, so that they can come again later if the branch is too busy (this is particularly the case at a campus university where the bank branches are on site). This means that subjects are more used to the idea of balking at banks, with their tolerance of queue sizes perhaps related to the sense of urgency for the business they wish to conduct. Thus the bank situation may lead to interesting balking behaviour. Finally, it is possible to split banking operations into separate services, each of which can have different queuing arrangements. This should provide more interest for the subjects using the simulator, and provide behaviour for a range of queuing situations.

6.2 The Bank Simulation Model

6.2.1 Overview of the Model

The Bank Simulation model concentrates on the front office activities of a hypothetical bank branch. Only those activities that directly relate to interacting with customers are modelled. Details of the implementation of the model are given in Appendix B. The description here deals more with the concepts and design of the model.

The Bank Simulation is built using Borland Pascal 7 (for DOS), with additional library routines to help with the process of model building that were developed by R. Hurron at Warwick Business School. The simulation engine uses the 3-phase method of approach.

The model was developed as an ordinary simulation of a bank. A user interface was then added to this that allowed a human user to make decisions on the actions of certain customers, and collected data on those decisions. The situation in the bank (number of counters open and queue sizes) at the point when a particular human controlled customer arrived was specified as a scenario. Each user of the simulation was shown a set of scenarios, these being stored on file and loaded in at the start of the session. The data collection interface, while requiring a significant amount of coding, was designed not to be integral to the central core of the simulation so that it could be easily removed once the data collection phase had been completed.



Figure 6.1 : Screen Shot of Bank Simulation

Figure 6.1 shows a screen shot of the Bank Simulation. This depicts four types of service counter available in the branch.

The Information desks are where customers come for advice, make arrangements for their accounts, etc. This is a multi-server, single queue system. The green desks are open, with staff (shown by an S) behind them, while the red desk is shut. The screen shot shows that only one desk is occupied by a customer.

The Transactions counters are where customers do everyday business in the bank, such as paying-in and withdrawing money. This is a multi-server, multi-queue system. Up to 5 counters can be open, although in the screen shot only 3 are open, with the closed counters indicated by a red bar.

The Business counters are for customers dealing with larger sums of money. This includes paying-in of shop takings, taking cash floats, and for anyone else with a business bank account. Business has a maximum of two counters open and like Transactions, it is a multi-server, multi-queue system.

The Currency counters deals with all issues concerning foreign currency and travellers' cheques. The counter system runs on the same basis as the Business counters.

Customers in the bank are shown by face symbols. Each is allocated a service to visit, and each joins queues on the basis of the shortest one (or random selection out of the shortest queues). The exception is the customer whos decisions are controlled by the human operator, which is shown by a red symbol of I, T, B or C, indicating the appropriate service required.

Among the other information shown on the screen is the clock in the top right corner indicating the current time of day in the simulation. In the top left corner are two time counters. Total time indicates how much time the human controlled customers have spent queuing in total across all the scenarios, while current time shows the time spent queuing in the current scenario. At the bottom of the screen is an information panel which provides the user with instructions on how to use the interface.

6.2.2 Bank Activities

Figure 6.2 shows an activity cycle diagram for ordinary customers in the bank system (those whos decisions are not under human control). The diagram is designed to retain the spatial relationships of the screen layout. The activities for most of the customers are fairly straight forward, except for Information customers, who might then go on

and use one of the other services and Transactions customers who might then go to information (shown by the dashed lines) .

Note from the activity cycle diagram that customers do not have balk or renege events, although these do exist for the human controlled customers. The model also contains staff entities which can be moved around the branch to take up different positions, however these are excluded at this stage since they are not of direct interest.

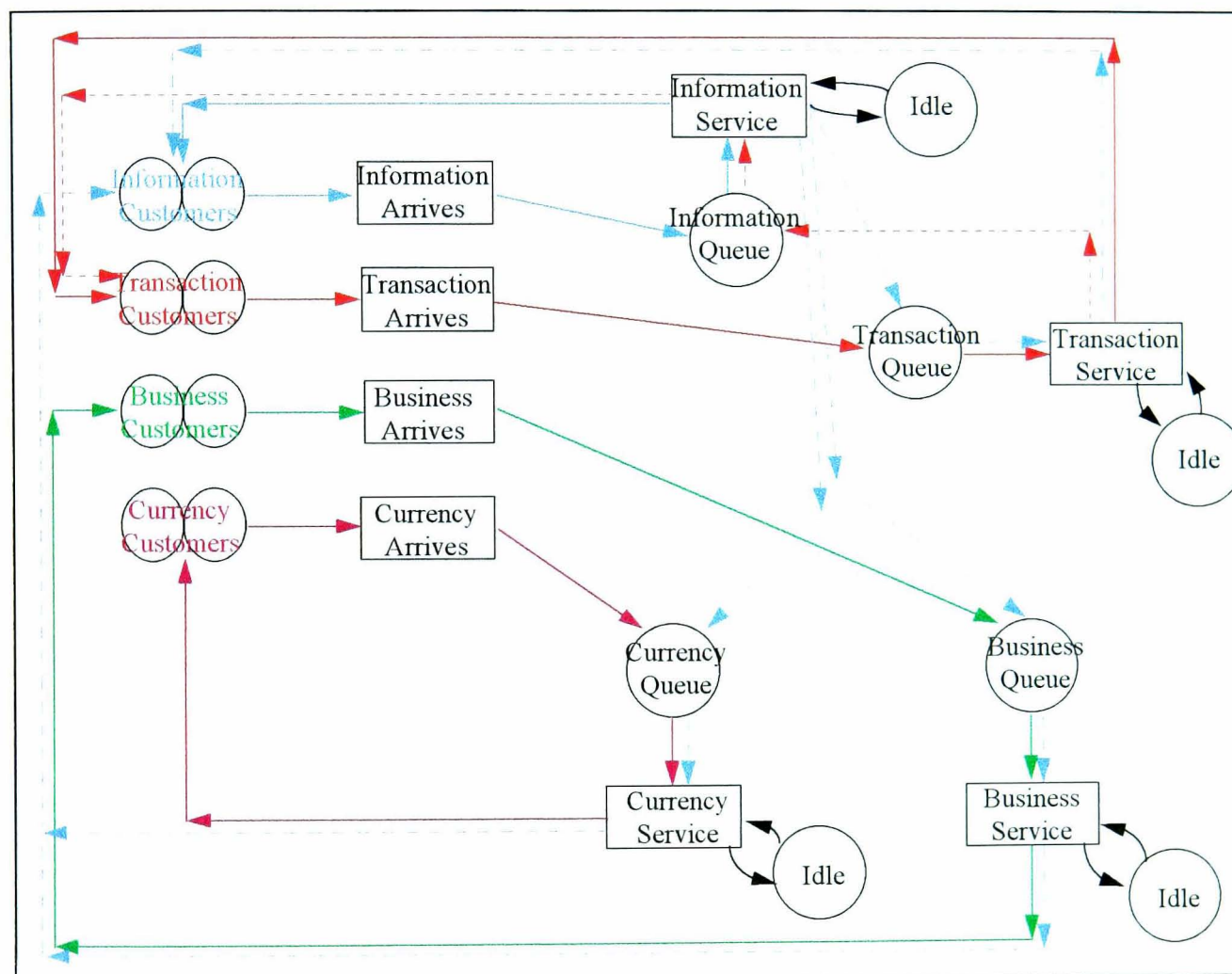


Figure 6.2 : Activity Cycle Diagram for Bank Customers

Customer arrival rates can be specified in a file that is loaded at run-time and uses a thinning approach to vary the arrival rates during the day. Details of the thinning, and the arrival rates used in the experiments are shown in Appendix B.

Figure 6.3 shows the activity cycle diagram for the human controlled (special) customers. These do not follow a random arrival pattern, but are specially generated

in each scenario. One special customer is generated for each scenario, to arrive at the time specified in the scenario file. These customers are represented by a single entity, but can follow a variety of routes around the system, depending partly on the attributes that are generated for them to represent the service that they require, and partly on the decisions of the human controller.

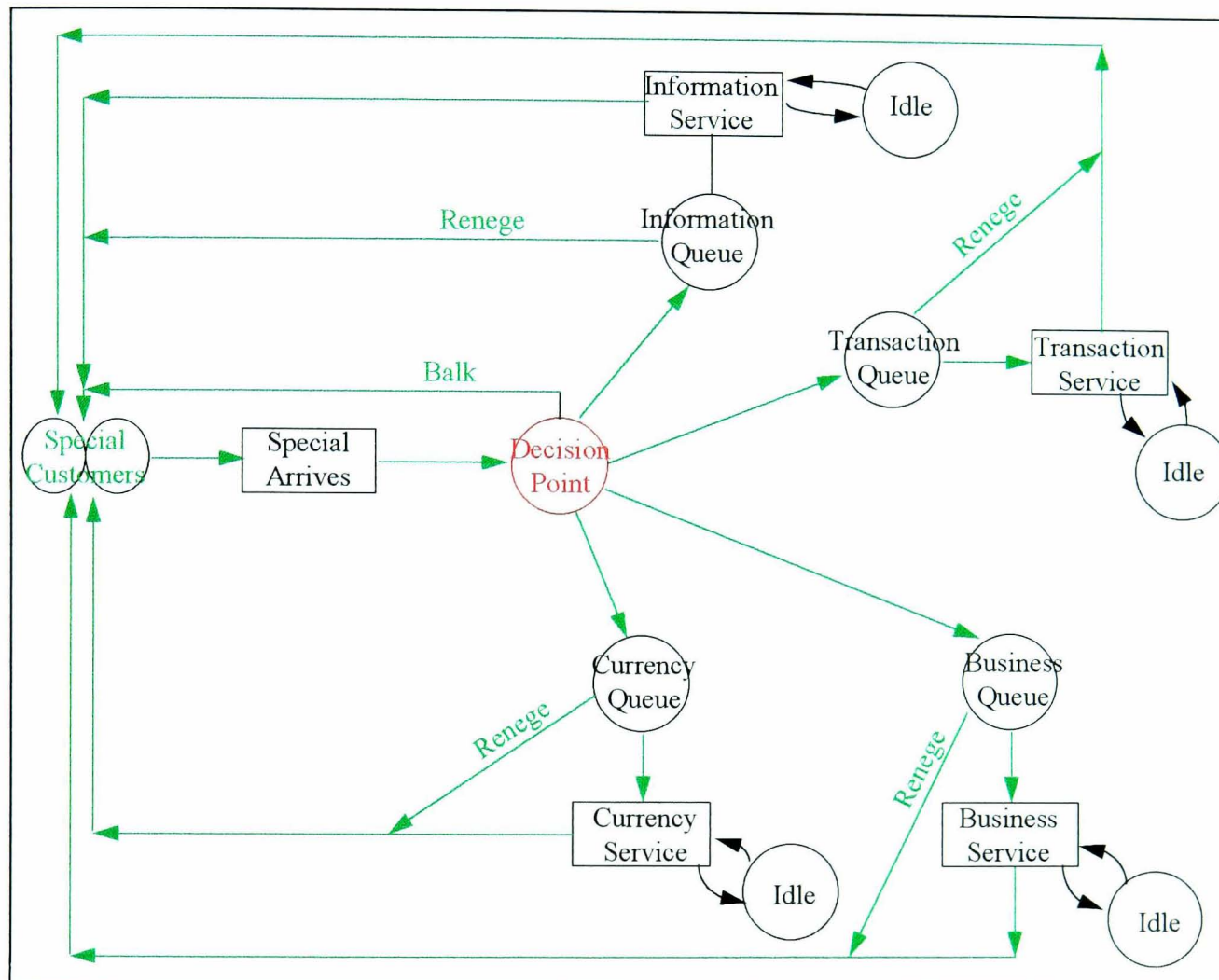


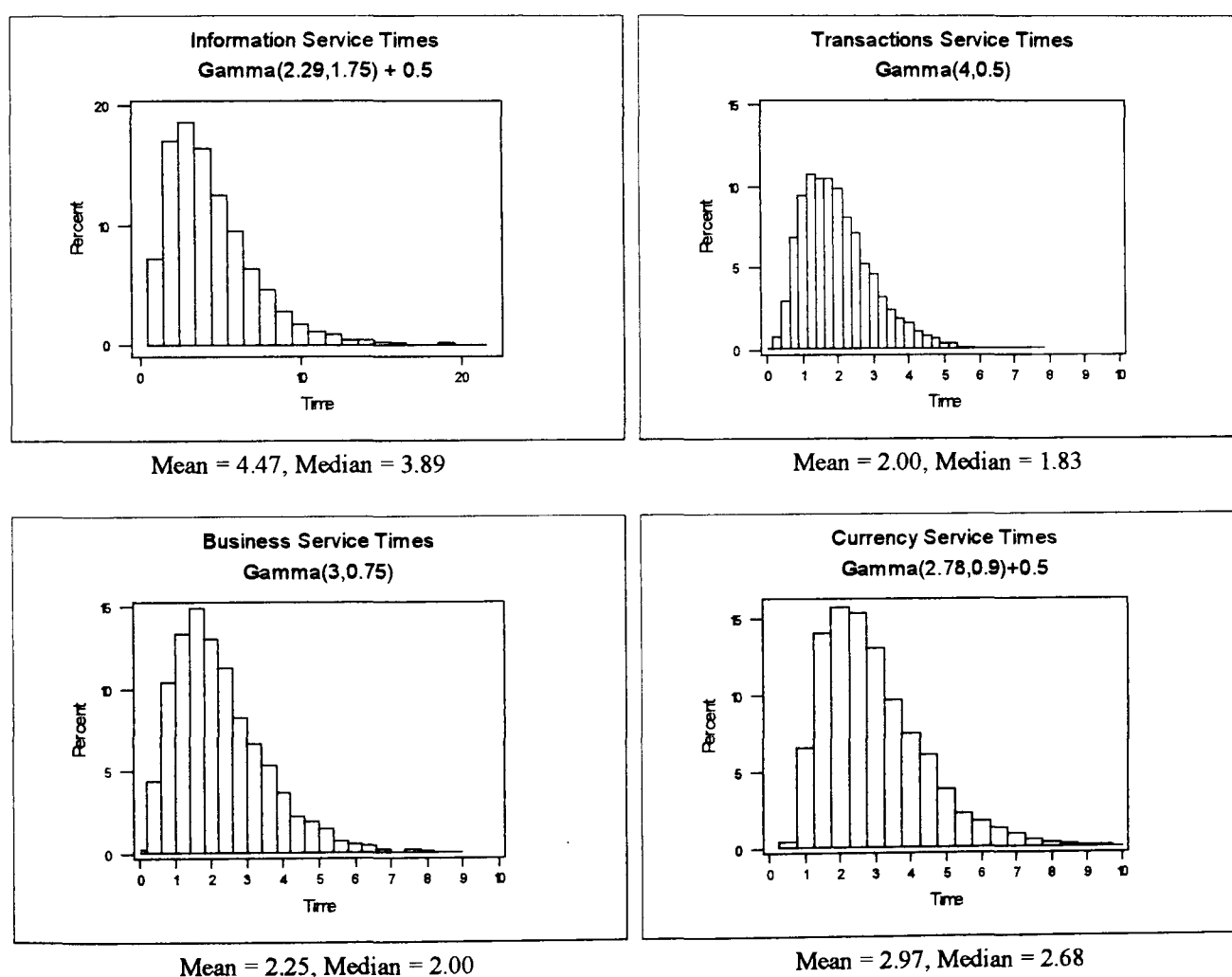
Figure 6.3 : Activity Cycle Diagram for Human Controlled Customers

Areas where customers might have to make decisions in the system (at least in the detail that it is being modelled), involve the choice of which queue to join when they first arrive, or to immediately leave the bank if their estimate of the queuing time is longer than they are prepared to wait. If the customer joins the queue, there are the options to swap queues or to give up on queuing and leave the bank.

The decision point shown in Figure 6.3 is a point where the simulation pauses for the human decision. It has no simulation time duration associated with it and so cannot

really be considered to be a true queue. At the decision point the human controller is allowed the opportunity to choose the queue to join for the appropriate service, or to leave the bank (balk). Having joined a queue, the human can at any time interrupt the simulation to change queues or leave the bank (renege). The mechanisms for doing this are described further in Section 6.2.3.

The service times for each type of counter are represented by Gamma distributions. The Gamma was chosen because of its skewed shape which, it was felt, would be appropriate for a typical bank service. The majority of services would be relatively straightforward, with a few taking considerably longer than average.



(Graphs and figures based on samples of 5000 data points)

Figure 6.4 : Service Distributions for Bank Simulation

Figure 6.4 shows the distribution for each of the counter services. The parameters were chosen through estimates from experiences in bank queues, and with a consideration of the variety of services that could be offered at each counter.

6.2.3 User Interactions

The human operator of the simulation is obliged to make decisions on the actions of the special customers. There are two points in the activity cycle of the special customer where human interaction can take place, as described below.

When the customer arrives, the simulation pauses for the human operator to make a decision for the actions of that customer. The symbol of the customer (I, T, B or C) represents the service required (Information, Transactions, Business, Currency). There is also a message display at the bottom of the screen that tells the user which service the customer requires, and also gives some instructions on what actions to take. The user has a choice of leaving the bank without undertaking the service (highlight the exit arrow), or of selecting one of the queues (highlight the counter) for the available counters of the service required.

The second point where the user can interact is while the customer is waiting in a queue for service. At any point the human controller can press the <space> bar to interrupt the simulation. There is a choice of leaving the bank, swapping to an alternative queue for the service required, or staying in the same queue.

The interface is similar for both the arrival and queuing cases, with the user selecting counters or to leave the bank by cycling through the options using the cursor keys, the option being highlighted by a green box. In the case of the information service, since it is a single queue system, there are only the options to join (or stay in) the queue or to leave the bank. The final selection is made by pressing the <Return> key. Figure 6.5 shows alternative options for an arriving Transactions customer.

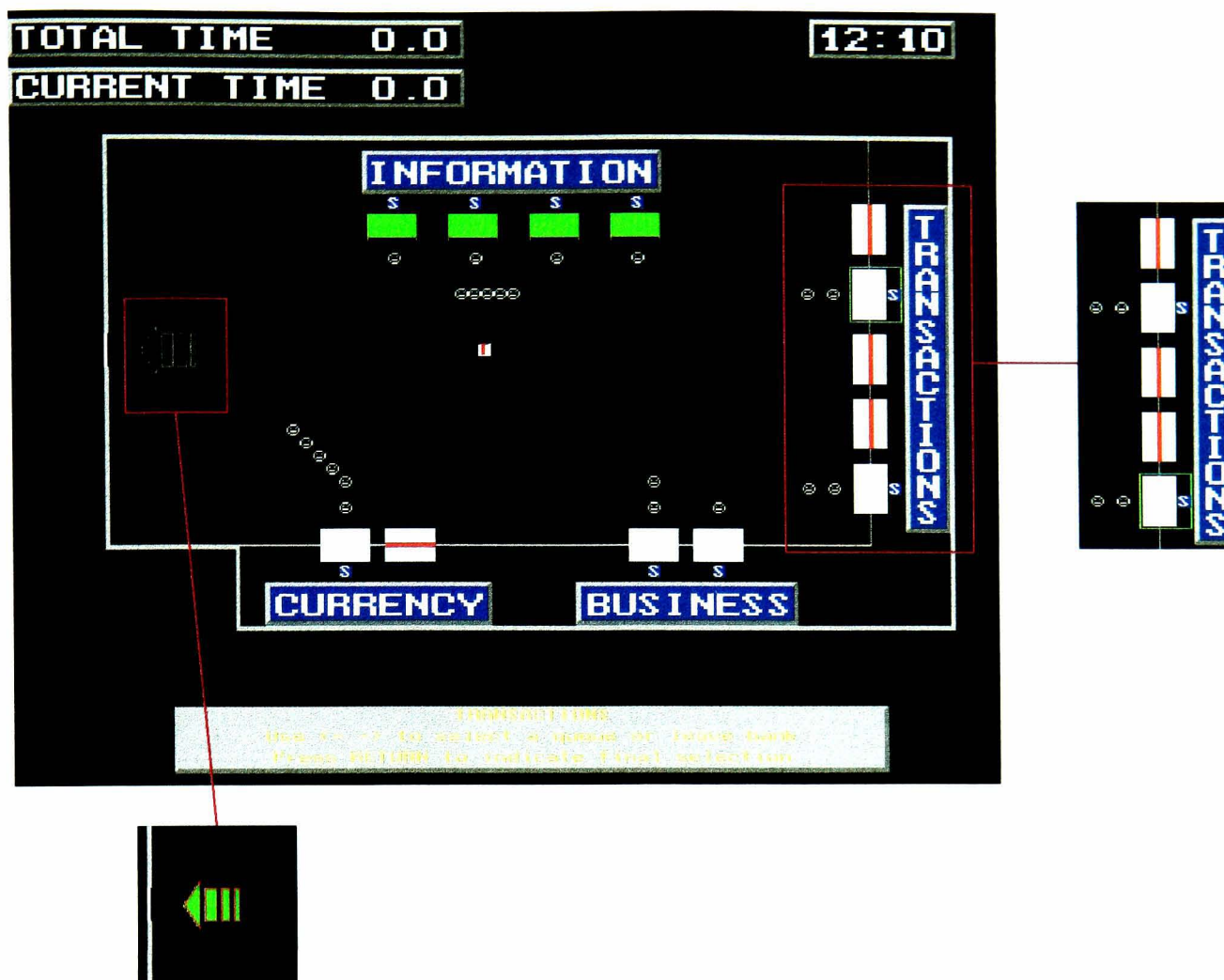


Figure 6.5 : User Selection Options for a Transaction Customer

The simulation runs faster than real time to speed up the data collection process. Due to the nature of the discrete event simulation engine, time does not advance in equal steps, but onto the time of the next event. However, a clock advance event was added to change the clock time in the top right hand corner of the screen every minute, so that the maximum time jump possible was 60 seconds (simulation time), and it contained a delay routine to pause the simulation briefly. This stopped simulation time passing very quickly without the user realising it.

The simulation framework allows control over scenarios that are shown to the participants. A real time simulation could have been used such that queuing in the bank for five minutes really did take five minutes for the user. However, this would have made the data collection process much slower, and there would have been the problem of the users getting very bored with the simulation.

6.2.4 Specifying Scenarios

The human users were shown a selection of scenarios each one involving a human controlled customer. The scenarios can be loaded in at the start of a session. The arrival time of the special customer is specified in the scenario, along with the service that they require, the number of counters open for each service, and the number of other customers using the same service. The queues for the other services are randomly generated. Time in the simulation is accelerated (with the graphics switched off) except when the special customer is involved in the queuing system.

The scenario files can be seen in Appendix C, and the choice and details of the scenarios are explained in more detail in Section 6.4.2 which describes the data collection.

6.3 The Pilot Study

6.3.1 Aims of the Pilot

The purpose of the pilot was mainly for testing the software and approach, but also to gain some insights into the customer decision making process to help with the main study. The pilot study involved collecting data from just one subject, developing decision making modules to replicate that person's decision making and implement the modules in the simulation model.

The subject was chosen as someone who was computer literate but had little experience of programming or simulation. This allowed a reasonable test for how easy it was to understand what was going on in the simulation, and the ease of use of the user interface. The pilot also offered the chance to test that the data collection framework was working properly and that the data collected was suitable for developing the decision making modules. Finally, implementing the decision making modules in the simulation allowed these to be prototyped, and to test that they would work within the simulation environment.

The pilot study will only be examined briefly, since most of the details are covered in depth in the description of the main study. However, the main observations and lessons gain from the pilot are described below.

6.3.2 Data Collection

The scenarios were chosen such that there was a reasonable spread of situations which would be shown to the subjects. The scenarios for the pilot study are shown in Appendix C. The scenarios were changed for the main study, mainly because the pilot showed the requirement of some longer queues than were used in the pilot. The total number of scenarios for the pilot was 58. This was felt to be too many for one sitting (taking approximately 40 minutes to run through them all), so these were split into two sets of scenarios. In the pilot, the same person was used for both sets of scenarios, but these were done in two separate sittings.

Even with the shorter sittings there was still the issue of keeping the interest of the subjects. It was decided that the simulation should take on aspects of a game, with a score, and that for the full study a prize should be offered to the person who got the best score. The score was determined by measuring the amount of time the human controlled customers spent queuing. On top of this there was an exit penalty which was added whenever a customer balked or reneged. If a customer balked right at the beginning of a scenario, the balk penalty would be added to their score, but they would have 0 queuing time. Thus if they thought the queuing time would be greater than the penalty it would be best to leave the bank. If the customer reneged after joining a queue, the balk penalty plus the time that they spent queuing would be added to the score. This is described further in the user instructions which can be found in Appendix D. In the pilot study the balk penalty was fixed at 6 minutes for all the scenarios, but it was decided to vary it for each scenario in the full study.

Data was collected for the arrival decisions and the queuing decisions, each set being placed in different files. The details of the data collection are described more fully in Section 6.4 for the main study. Brief details and observations are given here.

Upon arrival, data was stored for the type of service required, the number of open counters for that service, the size of each of the queues and the decision made. This remained the same for the full study. After a customer had joined a queue, data was collected when they changed queues or reneged. The data collected was the same as for arrival, with the addition of the position of the customer in the queue and the time that they had spent queuing. It was considered important to also have data for when the subject chose not to change queues, and so data was collected for every minute of simulation time. In the full study this was changed, so that data was recorded every time the queue state changed, since that provided a better dynamic picture of the speed of the change in queue sizes.

In addition to automated data collection, there was a questionnaire at the end of each session. The aim of this was to see if the subject could identify if there were any rules that they were using, and if so, how consistently they were applied. The questionnaire used in the pilot went on to be used in the full study with minor amendments. The questionnaire results, particularly when compared with the automated data collection showed that the subject could identify only very rough rules of thumb, and that these were not consistently applied in practice.

Another issue of interest was the degree to which the subject had learned to improve their decision making during the scenarios. In discussion, it appeared that there had been a period of learning initially, but little was gained after that. It was decided that a practice session should be added to the full study to allow the subjects to get used to the simulation and to develop some initial learning before any data was collected.

6.3.3 Modelling and Implementation

Much of the thinking for coding of the variables that was used in the full study was actually done for developing the models in the pilot study, and in fact was done even before this in deciding what data should be collected. Collecting data from only one subject did not allow the development of stochastic neural networks. However, ordinary deterministic neural network models were developed (though these did indicate areas of inconsistent decision making, even where only one subject was

involved). The development of these models tested that the type and format of the data was suitable for use with a neural network and also showed areas in which there were not sufficient examples to develop an effective neural network model.

For arrivals, once the inconsistencies had been (arbitrarily) removed, suitable neural network representations were found for the stay/balk decisions. Each service type was modelled with a separate neural network. For deciding which queue to join, a set of fairly clear rules could be identified and so a rule-based approach was used. Thus the arrival decision was split into two parts, a neural network stay/balk component, and a rule-based queue joining component.

Upon joining the queues, there were very few instances of renegeing on which to base a model. Queue changing was also quite rare, although reasonably consistent rules could be identified. For the full study, having more examples, a variety of decision makers, and collecting data when queue states changed (rather than using time intervals) it was felt that there would be more scope to use neural network models of queue changing.

Implementing the neural networks and rules into the simulation model was relatively unproblematic. The rules were simple enough that they could be easily coded into Pascal, while the neural network library which had been developed earlier allowed several different neural networks to be active at once in the simulation, and required very little coding in the main program.

6.4 Data Collection

6.4.1 Considerations in Setting Up the Data Collection

The aim of the data collection phase was to collect decision making data from a variety of subjects, so that a model containing variability in the decision making could be developed.

Since the data collection phase was time consuming, and required a number of willing participants, a key consideration in setting up the study was to gain as much structured

information as possible from the limited number of trials. To this end the scenarios specifying the situations that the subjects were asked to react to were specifically chosen to give a good spread of information.

There was an awareness that subjects were being asked to make decisions in an artificial environment. In this sense, they had no purpose in their decision making (since they did not really require any banking services) and would be missing factors external to the system that might affect decision making (such as the urgency of the business, time available and other tasks that needed to be done in that time).

It was important that the participants were given some sort of purpose for making their decisions. To this end, the bank simulation was turned into a game with points and a prize for the winner so that the participants felt that they were competing with others to produce the best score (and consequently win a prize). To facilitate the scoring and add a framework for the decision making, a system of penalties for exiting the bank without being served were introduced. The exit penalties introduced an external condition to the bank system for the decision making that could be thought of as representing the urgency of the business and amount of time available. Also the penalties gave a bench mark against which participants could gauge their decisions for deciding whether to join the queue or leave. The details of the penalties are described in Section 6.4.2.

Apart from the automatically collected data from the decisions, the opportunity was taken to also ask the participants questions about their decisions. This allowed a judgement of how well people could explain their reasoning, and could be compared against the decisions that they actually made. Also, it might allow the development of rules in situations where the decision making was easy to model with rules, or where the neural networks could not produce a satisfactory model of the decision making.

6.4.2 Setting Up Scenarios

Scenarios were used to specify the situations that the participants needed to make decisions for. It would have been possible to choose the situations randomly, say the

state of the system on the n th arrival, or the next arrival after some point in time. However, specifying the situations allowed a structure to be imposed on the data collection to make best use of the observations from a limited number of subjects.

Service Type	Counters Open	Number of Customers for Service					
Information	1	1	3	5	7		
	2	3	5	7	9		
	3	4	6	8	10		
	4	5	7	10	14		
Transactions	1	1	3	5	7	9	
	2	4	7	10	14		
	3	6	9	13	21		
	4	9	12	20	30		
	5	10	16	24	37		
Business	1	1	2	4	8		
	2	2	3	4	9	12	16
Currency	1	2	3	4	6		
	2	3	5	6	8	10	12

Table 6.1 : Summary of Scenarios for the Bank Simulation

Table 6.1 shows a summary of the complete set of scenarios for the Bank Simulation study. This was produced when devising the scenarios originally. The aim was to get a good spread of situations given a limited number of scenarios. In looking at the numbers, it should be remembered that the customers will be split into the appropriate number of queues (except for Information which has a single queue), and that some of the customers will be being served at the start of the scenario.

The scenarios were split between two scenario files randomly, with a particular participant being exposed to one of the files. The rest of the data in each scenario (time of day, data for the other service types) was generated randomly since it gives a background to the scenario but is not of direct relevance. In addition to the two scenario files, another practice file was created which contains 10 examples. These practice scenarios were shown to every participant to help them get used to the simulation before the data collection phase started. All the scenario files are shown in Appendix C.

6.4.3 Specifying Exit Penalties

Each scenario that a subject sees has an exit penalty associated with it. This is the queuing penalty time incurred when a customer does not wait to be served. The subject needs to estimate the queuing time and compare this against the exit penalty to see if it is worth staying.

In the pilot study, the same exit penalty was used throughout, however a variety of exit penalties were used in the full study. This was done to encourage a variety in the decision making, and to make the experience more interesting for each of the participants. A spread of penalties was used : 2, 5, 8, 11, 14 minutes to give a range from ‘a quick visit’ (2 minutes) to ‘urgent’ (14 minutes).

Each of the exit penalties was applied to each of the scenarios so that every 5 participants using the same scenario specification file would each see a different set of exit penalties for each scenario. The exit penalty applied to a particular scenario for a subject was determined using the simulation start time for the scenario and the subject identity number. The simulation start time for a scenario was randomly generated for the scenario data file, while the identity number is structured and unique for each participant. The scenario will always have the same simulation time, but different scenarios will have different starting times. Therefore, the simulation time acts as a starting point for determining the exit penalty which will differ between scenarios. The subject identity number is simply 1 for the first subject, 2 for the second, and so on. All odd numbered subjects were shown Scenario file 1, and even numbered subjects shown Scenario file 2. The exit penalties are stored as a list, and the item number of the list is incremented based on the subject identity (rotating the list so that incrementing past the last value gets back to the first value in the list).

The actual algorithm for determining the exit penalty (in Pascal format) is:

$$Index := 5 - (trunc(T) + i \text{ div } 2 + 1) \bmod 5$$

where T is the arrival time of the human controlled customer in the scenario, i is the identity number of the subject. *Index* will be an integer value from 1 to 5, and refers to the position in the list [2,5,8,11,14] for the exit penalty.

The exit penalty was prominently displayed throughout each scenario, as shown in Figure 6.6. This was put in as a large visual reminder for the participants to take account of the exit penalty in their decisions.

A further set of scenario files were produced which simply reversed the order of the scenarios, and were used for the second half of the study. This was done to try and reduce any bias in the results due to participants learning whilst using the simulation.

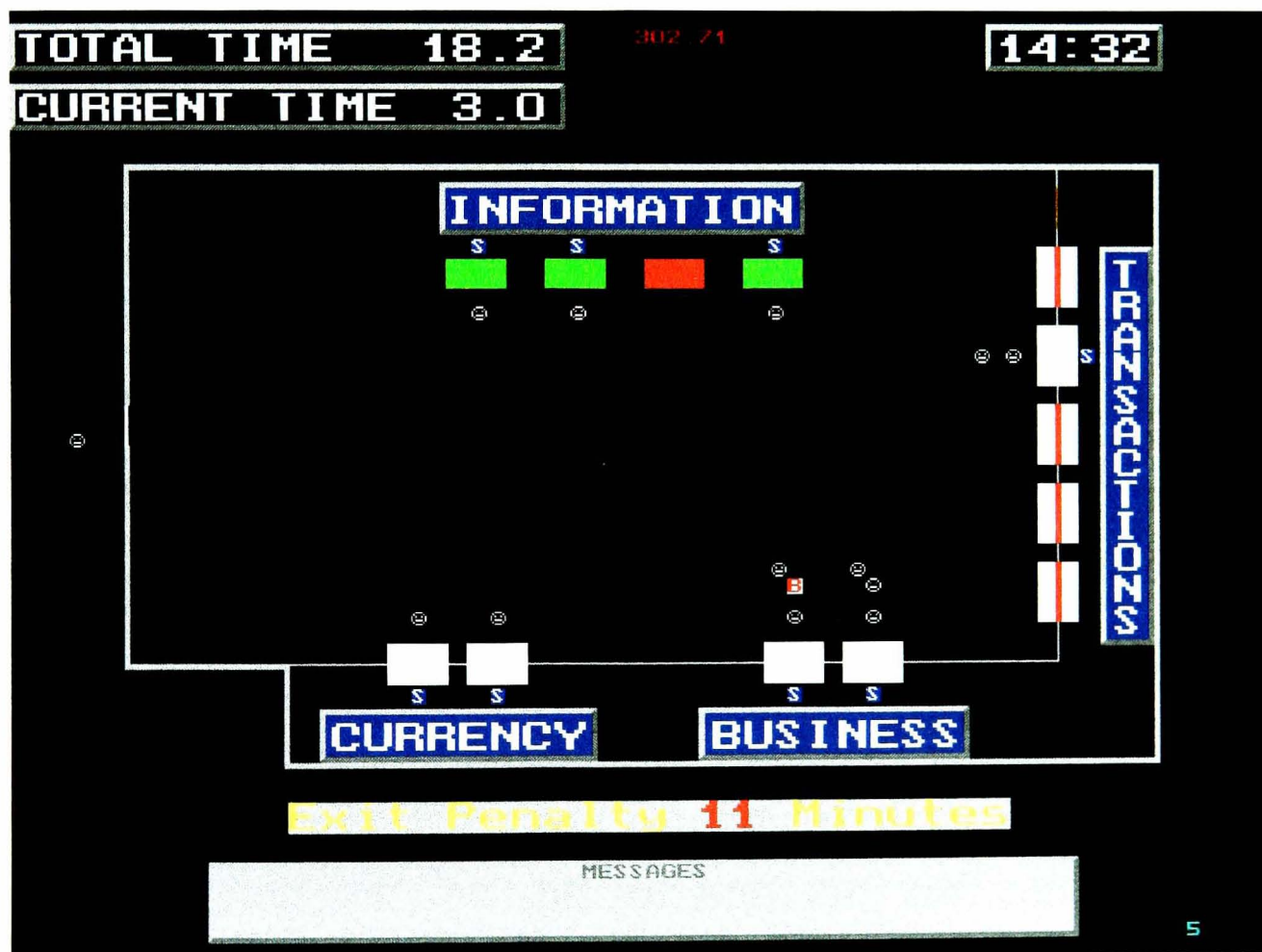


Figure 6.6 : Screen Shot of Bank Simulation Showing Exit Penalty

6.4.4 Recording Decision Data

Data on the decision making was collected for two main sets of decisions. Firstly there was the decision when the customer first arrives at the bank on whether to leave or stay, and if they stay, which queue to join. The other decision is when the customer

has joined a queue, and involves deciding whether they should swap queues, leave the bank or stay where they are. The data for the decisions were stored in separate data structures.

At the end of a session, the data was written to two files, one for the arrival decision data, and one for the in-queue decisions. The identity number of the subject was appended onto the file names for ease of reference.

6.4.4.1 Arrival Decision Data

The format of the data collected for the arrival decision is shown in Table 6.2. The first eight columns relate to the conditions under which the decision is being made, while the last records the actual decision. Specific notes and descriptions are given for each of the columns in the text below.

Column Number	1	2	3	4, 5, 6, 7, 8	9
Meaning	Exit Penalty	Service Required	N° of Open Counters	Queues for Counters	Outcome
Possible Values / Notes	2,5,8,11,14 Minutes	1=Information 2=Transactions 3=Business 4=Currency	0 - 5	Integer -1=Closed incl person being served	Queue N° : 1 - 5 0=Balk

Table 6.2 : Format of Data Collected for Arrival Decision

Column 1 : Exit Penalty

The exit penalty is recorded for administrative purposes, but is not actually required for building the decision making models.

Column 2 : Service Required

The service required specifies which of the four service types the customer requires. This is required because the service type is likely to affect the decisions made, and also affects how some of the following data is interpreted.

Column 3 : Number of Open Counters

The number of counters shows how many of the counters for that service were open. This is thought to be an important factor in determining whether a customer stays or leaves the bank.

Columns 4 to 8 : Queues for Counters

The next 5 columns give information about the state of each of the queues. Only the transactions service actually has a possible 5 queues. For the other services some of the columns are redundant, but the value -1 (closed) is recorded for the purpose of data formatting. The queue size includes the person who is actually being served as well as those waiting to be served. The Information counter is a special case since it only has one queue. Here the data is still recorded for each counter that is open, involving the size of the single queue plus the person being served. If there is no-one in the single queue, this record indicates which counters are involved in serving someone, and which are free.

Column 9 : Outcome

The outcome records the decision that was made. A value of 0 indicates that the customer left the bank without joining a queue, otherwise the number for the queue that they joined is recorded. The queues for each service type are numbered separately: Information has only one queue, so that is 1, Transactions has queues 1-5, while Business and Currency have queues 1-2. All of the queues are numbered in a clockwise order. In retrospect, having only a number 1 for the Information queue is limiting, since if several Information counters were free it would not indicate which counter the customer went to. However, the scenarios only covered the situation where all the open counters are occupied and so the limitation did not affect the results.

Another factor which could affect the decision making, but which was not included in the model (and therefore no data was collected for it), is the nature of the business that the people who being served are involved in. Some business may be easily observed by other customers as being time consuming, and so customers might avoid that queue.

Several participants mentioned this factor, and any later study might include such a facility.

6.4.4.2 In Queue Decision Data

Data was also collected on decisions made once a customer had joined a queue. While in the queue, the customer can swap queues or decide to leave the bank (renege). While the arrival decision can only occur at a specific point in time (when the customer first enters the bank), the decision to change queues or leave can occur at any time while the customer is in the queue, adding a temporal dimension to the decision making. Data was recorded every time the state of the customer in the system changed, that is, when they swapped queues, left the bank, or their position in the queue changed (through the person at the front of the queue leaving after being served). This provided information on when the customer made a positive decision and gave snapshots of situations when they decided to stay where they were.

Column Number	1	2	3	4, 5, 6, 7, 8
Meaning	Exit Penalty	Service Required	N° of Open Counters	Queues for Counters
Possible Values / Notes	2,5,8,11,14 Minutes	1=Information 2=Transactions 3=Business 4=Currency	0 - 5	Integer -1=Closed incl person being served
Column Number	9	10	11	12
Meaning	Current Queue	Position in Queue	Time Spent Queuing	Outcome
Possible Values / Notes	1 - 5	Integer 0=Being Served	Minutes	Queue N° : 1 - 5 0=Renege

Table 6.3 : Format of Data Collected for In-Queue Decision

The format for the data collection is shown in Table 6.3. The first 8 columns are the same as for the arrival decision, while the other columns are described in more detail below.

Column 9 : Current Queue

The queue that the customer is currently in is recorded so that the rest of the data can be considered in context, and is compared against the decision outcome.

Column 10 : Position in Queue

The position of the customer in their queue is important, since their position relative to the length of the other queues is likely to be a significant factor in deciding when to swap queues.

Column 11 : Time in Queue

Since the decision making has a temporal dimension, particularly for reneging, this is recorded as the length of time that the customer has spent queuing.

Column 12 : Outcome

The outcome record is much the same as for the arrival decision. The queue number that they are in after the decision has been made is recorded. This may be the same as the queue noted in column 9, indicating that they have not moved, a different queue number showing that they have swapped, or 0 indicating that the customer has reneged.

Some example data sets for both arrival and in-queue decisions can be found in Appendix E.

6.4.5 Conducting the Data Collection Sessions

The sessions were conducted with subjects one at a time, and was supervised. To encourage people to participate, two prizes of £40 were offered to the people with the best scores (a prize for each scenario file). A session typically lasted between 25 and 40 minutes depending on the speed with which the subject understood what they needed to do, the speed of their decision making, and the amount of discussion in the questionnaire at the end.

Decision making data was collected from 20 subjects, with half the subjects using the scenario 1 file, and the other half using the scenario 2 file. With 5 sets of balk penalty values it was felt that the total number of subjects should be a multiple of 10 so as to maintain a balance. More subjects mean more data for developing the models, but this

was constrained by time and the number of willing participants. Subjects were restricted to only one session with the simulation model (although several did ask to have another go), and the person who was involved in the pilot study was not permitted to be involved in the main study.

A Bank simulation gaming session was conducted as follows :

1. Instructions

The participants were given written instructions describing the purpose of the study, aims of the simulation and the keyboard controls for making decisions. The instruction document is shown in Appendix D. The participants were able to ask any additional questions to clarify the instructions, but were not given any details on the dynamics of the simulation (such as service times). Participants were given the choice of controlling the simulation directly, or to verbally express their instructions and have these entered for them. The split on this was approximately 50-50. No indication was given at any point regarding how many scenarios the subjects would be shown, or of what the best scores were so far, since these might effect the manner in which the decisions were made towards the end of the session.

2. Short Practice Session

All of the subjects were allowed a practice session with a scenario file containing 10 scenarios. They were told that the score on the practice would not be recorded, and that they should use the practice as an opportunity to get used to the bank environment and the simulation controls.

3. Data Collection Session

Each subject used the Bank simulation with one of the two scenario files, and data was collected on their decision making. During this phase no instructions or advice was offered except for responses to questions about the simulation controls.

4. Questionnaire

At the end of the data collection session, participants were asked a number of questions about their experience, and about their decision making. This was done using a questionnaire.

In most respects, the actual scores that people achieved were of little interest as far as modelling the decision making was concerned. It was however interesting to compare the decision making styles of those who got good and those who got poor scores. The participants were not really competing on a level playing field, since the sequence of exit penalties that they received varied, although there was still quite high variation in the scores for people who were given exactly the same scenarios with the same exit penalties.

6.4.6 The Decision Making Questionnaire

The questionnaire used at the end of each session can be seen in Appendix F. The aim of the questionnaire was to determine if the subjects could identify any specific rules in their decision making. The responses allowed a judgement of the ease with which people could explain their reasoning, and also could be compared with the data that was collected for the session. Secondly, the responses helped to determine the main criteria for making decisions and to determine rules in areas where a rule-based approach could be used, either instead of, or in conjunction with, a neural network approach.

The participants were asked questions from the questionnaire, rather than actually having to fill it out for themselves. Care was taken not to put words into their mouths, but in some cases clarification was sought from the participants on a point that they were making.

The questionnaire contains a cover sheet to write down information on the subject number, scenario, data and also the score achieved. The participant's name was also

added for the purpose of awarding the prizes. The questionnaire itself comprises of six questions which will be discussed below.

Question 1

This was a tick box question asking participants to rate the difficulty of deciding whether to stay in the branch or to leave, with a range from 1 (easy) to 5 (hard). This asked for an overall assessment, and for each of the individual service types. While the answers were highly subjective, it did allow a comparison of the perceived difficulty for the different services.

Question 2

This asked for any specific rules that were used in deciding whether to stay in the branch or leave, and whether the participant actually stuck with the rules. This was done for each of the service types. In some cases, where participants were struggling to come up with any rules, they were prompted for the factors which they thought were important in making the decision.

Question 3

The participants were asked if they were generally happy to stay in the queue that they had first joined, and why (or why not). This behaviour was usually evident from watching the session, but it was interesting to hear people's reasoning.

Question 4

This question built on the answer from 3, to ask about the criteria for deciding when to change queues. This was asked separately for each of the service types, though the Information service was excluded because queue changing is not possible. There was an additional question that asked if the participant thought their decisions to stay in or change queues had generally paid off. This was a subjective question but did give an insight as to whether they were generally optimistic or pessimistic about the results of their decisions.

Question 5

The participants were asked whether they had left or thought of leaving the bank after joining a queue. This gave some insight into the attitude of actually leaving a queue after some time had been invested in it. There were supplementary questions about the factors that led to people thinking about leaving, and what actually resolved the decision. Since reneging has a temporal dimension to it, it was felt to be important to find out not only what decisions were made, but also to identify when people starting thinking about it.

Question 6

This asked about how the participants felt their performance had changed during the session, with options of *worse*, *the same*, or *better*. Responses such as ‘a bit better’ were represented as being between two choices. A follow up question asked participants to explain the reasoning behind their statement. The question gave some idea of the degree of learning that had gone on during the data collection session. There was a level of subjectivity in the answers, particularly if the person could identify some unfortunate decisions near the beginning or end of the session. While the question does not help in developing the decision making models, it gives some insights into the decision making process. Also the results of the question lead to reversing the order of the scenarios half way through the study to balance any biases in the responses.

Some of the insights gained from the questionnaires will be discussed in Section 6.5 where they affected the model building.

6.5 Data Analysis and Model Building

6.5.1 Organising the Analysis

The task of modelling the decision making of customers in the Bank simulation involves looking at two main areas of decision making, those made when the customer first enters the bank (or goes on to choose another service) and those when the customers has joined a queue.

As discussed in Section 5.1.2, making the most of the data available is likely to require splitting the decisions up into several parts, and generalising some of the decision factors. This is the first stage of the analysis, and was actually done before the data collection since it is required for determining what data is needed, and how much. However, for ease of understanding the process, that stage is described here. Also, some amendments might be required based on observations from the data that has been collected.

Data analysis was done for each sub-decision. This generated an increased understanding of the nature of the data and the decision making prior to model building. It was used in determining which modelling approach was most appropriate, and also for deciding how reliable the models might be. Gaps in the data and insufficient data could be spotted at this stage.

Model building then took place for all the sub-decisions. Each sub-decision can be handled separately, but an awareness was required of how the sub-decisions fitted together eventually. Analysis of the decision making models was done at this stage, before implementation in the simulation model.

The neural network models were developed for a number of network architectures involving different numbers of hidden nodes. Each model was run for 50,000 iterations, and the minimum fitting error obtained so far in the training process was recorded. The networks were stopped at the point after the 50,000th iteration where the current fitting error was less than or equal to the recorded minimum. This was done to ensure that the network was not saved at a point where the error rate had temporarily moved away from the minimum. It was noted during training that for all the models very little improvement in the error rate was occurring well before the 50,000th iteration, and so that was seen as a safe cut off point.

Validation of the models is difficult when the amount of data available for modelling is limited. This is a problem with all models of decision making, and particularly so when variation in decisions is modelled. All of the data was required for creating the

models, leaving none for validation. The stochastic nature of the models means that it is not possible to just remove a few data values for validation, instead large amounts of validation data would be required. However, models can at least be checked for internal consistency and stability.

In the end, the validation involved determining whether the models looked sensible and credible. This required a subjective judgement on the results of the decision making models. Input values could be generated and used with the trained networks to produce the outputs, and so the response of the network to different situations could be visualised. Without large amounts of data available, or further data collection, no more rigorous approach could be used. This is particularly an issue where different network architectures can produce different models. The issue of validation is discussed in more depth in Section 6.7.

To help in the process of mapping the sub-decisions and to record details of the modelling methods used, a Decision Mapping Form was created. This allows a diagrammatic representation of the relationships between sub-decisions, a description of the decision problem, the decision outcome and criteria, and details of the modelling approach. The use of the forms helped to document and record the structure and detail of the decision making models. The relationships between the sub-decisions, the modelling approaches, and the type and format of data needed to use the models can be seen. The Decision Mapping Forms for the development of the decision making models in the Bank Simulation can be found in Appendix G.

6.5.2 Identifying Sub-decisions and Dimension Reduction

Customer decisions can be identified as taking place in one of two areas, prior to queuing, and during queuing. Within these there are further sub-decisions which can help to reduce the dimensionality of the decision making models. It was also necessary to consider data transformations that might also reduce the dimensionality of the problem without compromising the accuracy of the model.

- **Service Types**

For both sets of decisions, there is an immediate form of sub-decision which relates to the type of service required. The queuing situations for the Information counters and the Transaction counters are very different, and so were considered as separate decision making models. The queuing for the Business and Currency counters are similar in terms of structure, and also have similar, but not identical, service times (see Figure 6.4). However, the perception of Business and Currency services by customers may well be different. These were treated as separate models, although a combined model was considered.

- **Balk / Join Queue Decisions**

The decision when the customer enters the bank is whether they balk, or join a particular queue. This can be split into two clear sub-decisions, that of whether to wait for the service or to balk, and then if they decide to wait, the decision of which queue to join. This split is applicable to all the services except Information which does not require a decision about the queue to join. For the stay/balk decision it allows the consideration of the number of counters open, and the shortest queue length, rather than the more specific situation of the pattern of queues. The pattern of queues must be considered for the decision of which queue to join, although this is fairly straight forward for all the services except Transactions.

- **Swap Queues / Renege Decisions**

The other main customer decisions are those that made after the queue has been joined. The decision to swap queues, or renege are quite different decisions that are likely to be sparked off by quite different criteria. Therefore these were considered as two separate decisions.

6.5.3 Overview of Decision Upon Arrival at the Bank

As noted in Section 6.5.2, the decision upon arriving at the bank can be considered separately for each service type, and can be split into two sub-decisions, firstly

deciding whether to leave the bank or wait for the service, and secondly, if the customer decides to wait, which queue to join.

Figure 6.7 shows an overview of the decision making when the customer first arrives at the bank. Decisions are represented by boxes, while actions are represented by circles. The Information service does not have a further sub-decision for which queue to join, since it employs a single queue system. The queuing sub-decisions for Transactions and Business have been left off the diagram for clarity, but follow the same structure as Currency, except that Transactions has a choice of up to 5 queues.

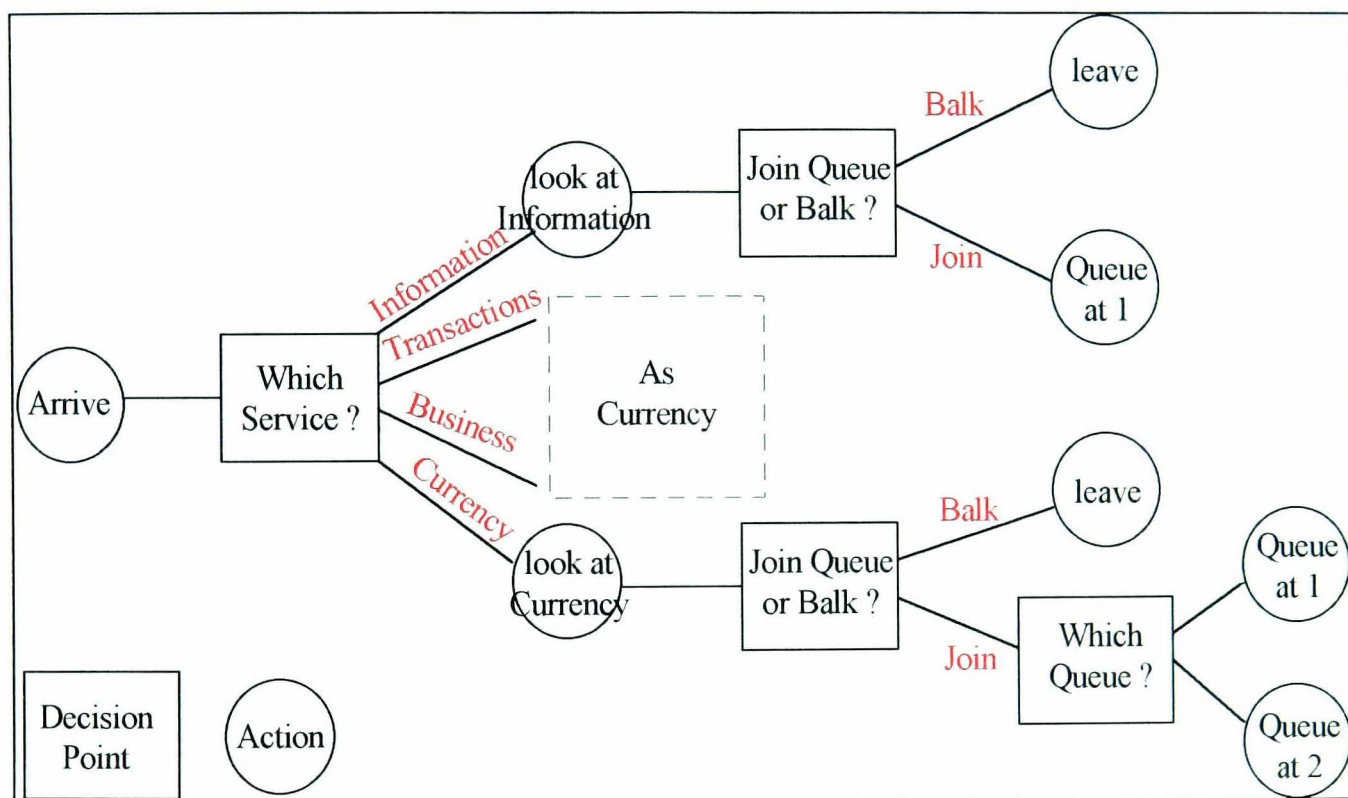


Figure 6.7 : Overview of Decision upon Arrival

The analysis and model building will be described for each of the sub-models separately. Appendix G contains summary sheets for all of the sub-models that were created during the model building stage to document the organisation and key features of the models.

6.5.4 ‘Which Service ?’ Decision

The decision of which service the customer requires is a trivial one. For the purposes of the simulation it is assumed that the customer comes into the bank requiring a

particular service. Some customers may require several services, but this is determined after the previous service has finished, with the customer dealing with the primary service first. Therefore all the criteria are external to the bank. Arrivals of customers for each type of service are generated by a different arrival process. Thus the customer type determines the service that is required.

6.5.5 Queuing Decision Upon Arrival at the Bank : 'Join or Balk ?'

Upon arrival at the bank, after deciding which service is required, a decision is made about whether to stay in the bank and join the service, or whether to leave the bank (balk). The following section outlines the models developed for each of the services to represent this decision. A similar approach is used for each service type, so only the development for the Information counters will be described in full detail, with summaries given for the other services.

6.5.5.1 Information : Join or Balk

The information service uses a multi-server single queue system. Therefore there is no decision required for which queue to join. Instead, the decision is only whether to stay in the bank or leave.

Counters Open	Size of Queue	% Joining
1	0	100
1	2	70
1	4	20
1	6	0
2	1	100
2	3	80
2	5	70
2	7	20
3	1	100
3	3	80
3	5	30
3	7	60
4	1	90
4	3	80
4	6	50
4	10	20

Table 6.4 : Information Data showing Percentage of Customers Staying

The data collection stage had 16 scenarios that involved the Information service, for each of which there were observations from 10 different subjects. Table 6.4 shows each of the scenarios with the percentage of subjects who joined the queue. Note that the size of the queue refers only to the number of people in the single queue, excluding those people who are being served at the counters.

One would expect the proportion of customers staying to decrease as the size of the queues increases. On the whole this can be observed in Table 6.4, although the results for 3 counters with queues of 5 and 7 people are counter to this. Also it is expected that customers would be more tolerant to longer queues if more counters are open. This effect can be seen, except for the case of 4 counters with only 1 person in the queue.

A drawback with the data is that it is only for the case of 1 counter open that any scenario has results where 0% of customers joined the queue. This makes determining the tail of the decision distribution difficult.

Looking at the comments in the questionnaire on the decision, most people indicated the importance of both the queue length and the number of desks open in their decision making. A few people used estimates of the average service time per customer (ranging from 1.5 to 3 minutes), others stated a few coarse rules, although there was little consistency between subjects. Most of the subjects admitted using an intuitive approach to the decision and having no clear set of rules.

The decision making process obviously does include stochastic elements (since there is a variety in the decision making), and involves interpolation between the number of counters open and the sizes of queues. This suggests the use of a stochastic neural network model, rather than a rule-based or ordinary random sampling approach.

The neural network model has to cope with some of the inconsistencies in the data. In particular it was hoped that the effects of the data with 1, 2 and 4 desks open would

outweigh the aberrant data for 3 desks. Also the lack of data for the tail of the distribution has to be dealt with.

At this stage it was not clear which type of neural network model would be best. The analysis in Chapter 4 suggested that using a probability variable as an input (INVAR) is likely to produce the best results, unless very accurate empirical probability distributions could be created. However, there is also a case for allowing the network to determine its own probabilities. At this stage of the investigation it was useful to compare all of the approaches outlined in Section 4.4 since the nature of the data is slightly different to that used in the tests using artificial data.

The data was prepared for all three neural network approaches. In each case there were two decision attributes, a) the number of desks (scaling from 1 to 4), and b) the size of the single queue (scaling from 0 to 15). There were two outputs, a) staying in the bank, and b) balking. The scaling of the number of desks covers the possible range of values. The queue size could in theory be infinite, but is unlikely to be above 15, and certainly no training data has a value above 10. Any queue sizes above 15 in the subsequent use the model will result in scaled values above 1, but that does not cause a problem for the network. The output target variables will always have values in the range 0 to 1 and so do not need scaling.

The process of preparing the data sets and analysing the models for each of the approaches will be looked at in detail for the Information service, but in less detail for the other services.

Using a Probability Variable as an Input (INVAR)

For the approach using a probability input variable (INVAR), empirical probabilities were generated for a third input variable. The different numbers of counters open formed a set of groups, and the different queue sizes formed another set of groups. Data points were divided up according the group pairs (for example, 3 counters, queue size of 6). As Table 6.4 shows, data was available for 16 of these group pairs, each having 10 data points, forming a data set of 160 examples. Within each group pair, the

‘stay in bank’ output (taking the value 0 or 1) was sorted in descending order, and the empirical probabilities applied (taking the values 0.05, 0.15, .. ,0.95).

A number of networks with different sizes of hidden layer were trained with this data, and the resulting networks were tested on a data set containing all possible combinations of number of desks open (1..4) and queue sizes (0..15). For each combination, probability values of 0.005, 0.015, .. ,0.995 were used to generate results from the network. The actual decision was made on the basis of the output with the highest value. The probability of a customer staying for a particular pair of input criteria is determined by the proportion of probability input values for which a stay result was achieved. The results for the different network sizes are shown in Appendix H1.

Figure 6.8 shows graphs of the results for the network with what are thought to be the best features based on an understanding of the decision process. The graphs show the probability of a customer staying in the bank to join the Information queue for different number of counters open and queue sizes. The actual probabilities represent the data points that the network was trained on.

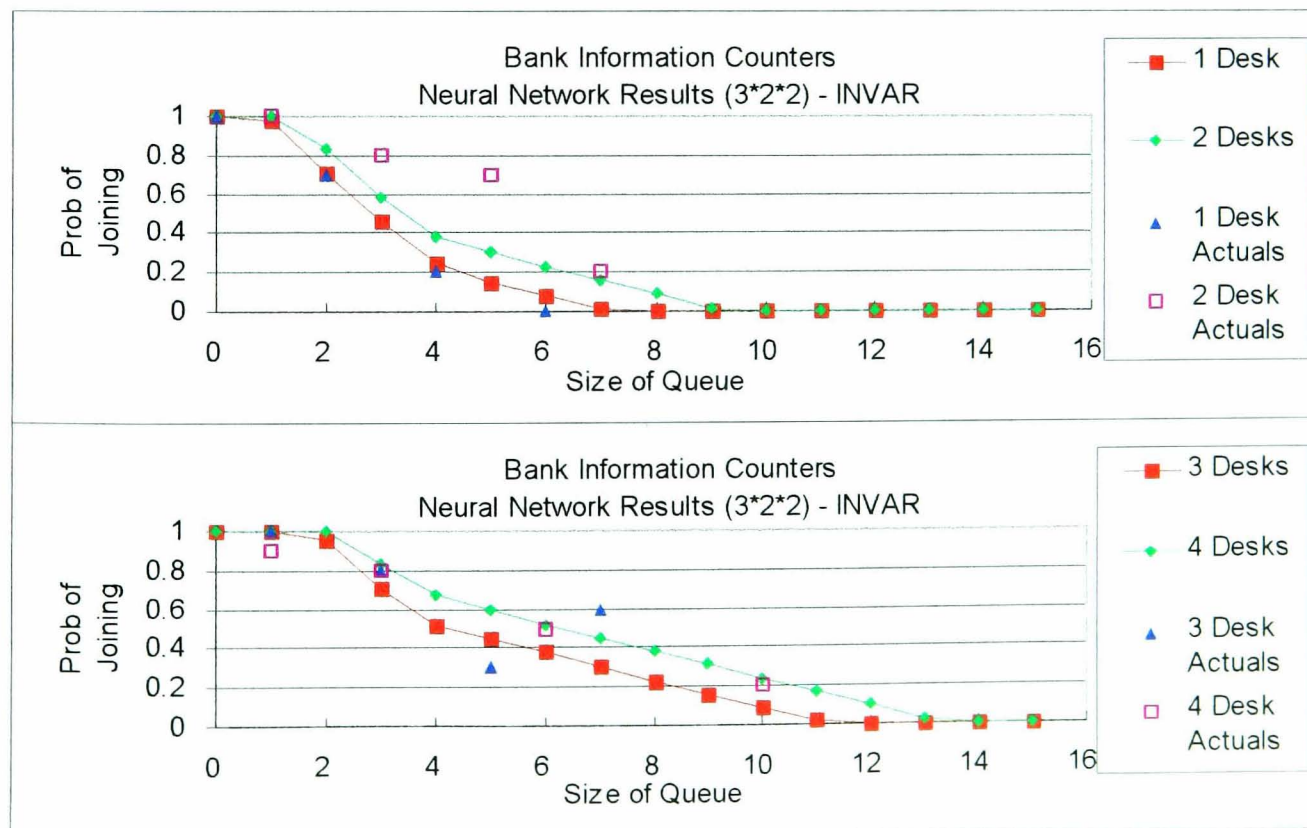


Figure 6.8 : Information Test Results for the INVAR Network

The graphs clearly show some of the inconsistencies in the data, and the way that the network has dealt with them. In this network, the effect of the number of counters open has had a strong impact on the model. The relationships shown between the different numbers of counters follow a pattern that is intuitively plausible. The strength of these relationships override a number of the inconsistencies in the data. The network also models a set of reasonable tails to the distribution, although the fact that they are closed could be argued as being not entirely desirable. Thus there is a point where no customers will join, rather than there being a very low probability of joining.

By way of comparison, the networks with larger numbers of hidden nodes ($3 \times 4 \times 2$ and $3 \times 3 \times 2$) were more affected by the inconsistent nature of the data. In the case of 2 counters open, the model tended to bulge more than expected in the middle, much as shown for the OUTDATA model in Figure 6.9. For three counters, the model is not badly affected by the highly inconsistent data for queue sizes of 5 and 7, but it does have the effect of suggesting that people are slightly more likely to stay if 3 rather than 4 counters are open, a result that is counter intuitive. Also the tails of the distribution tend to be rather too short. For the network with fewer hidden nodes ($3 \times 1 \times 2$), the fit is fairly linear in nature, exhibiting the right sort of relationship between the number of counters, but having tails that are too short.

Representing Membership Probabilities as Outputs in the Training Data (OUTDATA)

The preparation of the data set used the same groupings as for the INVAR method. The proportions of subjects making the choice to stay in the bank or leave were calculated for each grouping and resulted in 16 data points.

The data was used on networks with a variety of numbers of hidden nodes and tested with data on every combination of numbers of counters (1..4) and queue sizes (0..15). The results for each of the networks are shown in Appendix H1. Note that in all cases the output pairs produced by the network summed to 1 (to at least 4 decimal places), so no rescaling of the probabilities was necessary.

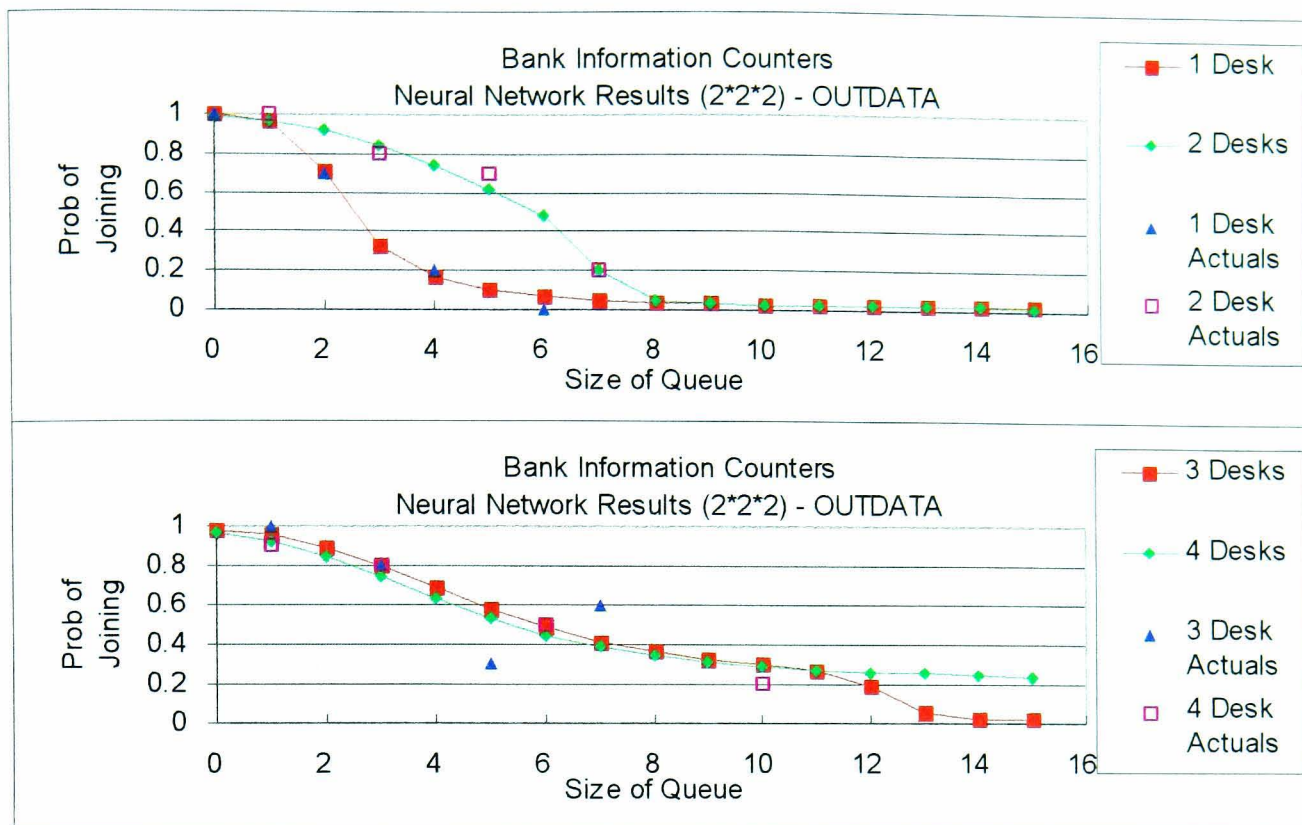


Figure 6.9 : Information Test Results for the OUTDATA Network

The graphs in Figure 6.9 show the fit of the network to the test data, and the points of the training data. The 1 desk case has a reasonable shape, 2 desks follows the data reasonably well, but has a bulge that contradicts the shape of models for the other numbers of desks. The 3 desk model copes fairly well with the contradicting data, but suggests that customers are more likely to stay if there are 3 rather than 4 desks, which is counter intuitive. All except the 4 desk case have tails with a good shape, tailing off towards, but not quite reaching 0 probability. For 4 desks, the tails never falls below 0.2.

The networks with larger numbers of hidden nodes coped poorly with the inconsistent data for 3 desks, with the probabilities of staying in the bank increasing after a queue size of 7 people after that point. The 2*3*2 network suffered from this, but did have smoother tails than the network shown in Figure 6.9. The smaller network with 1 hidden node was largely a linear fit, with short tails.

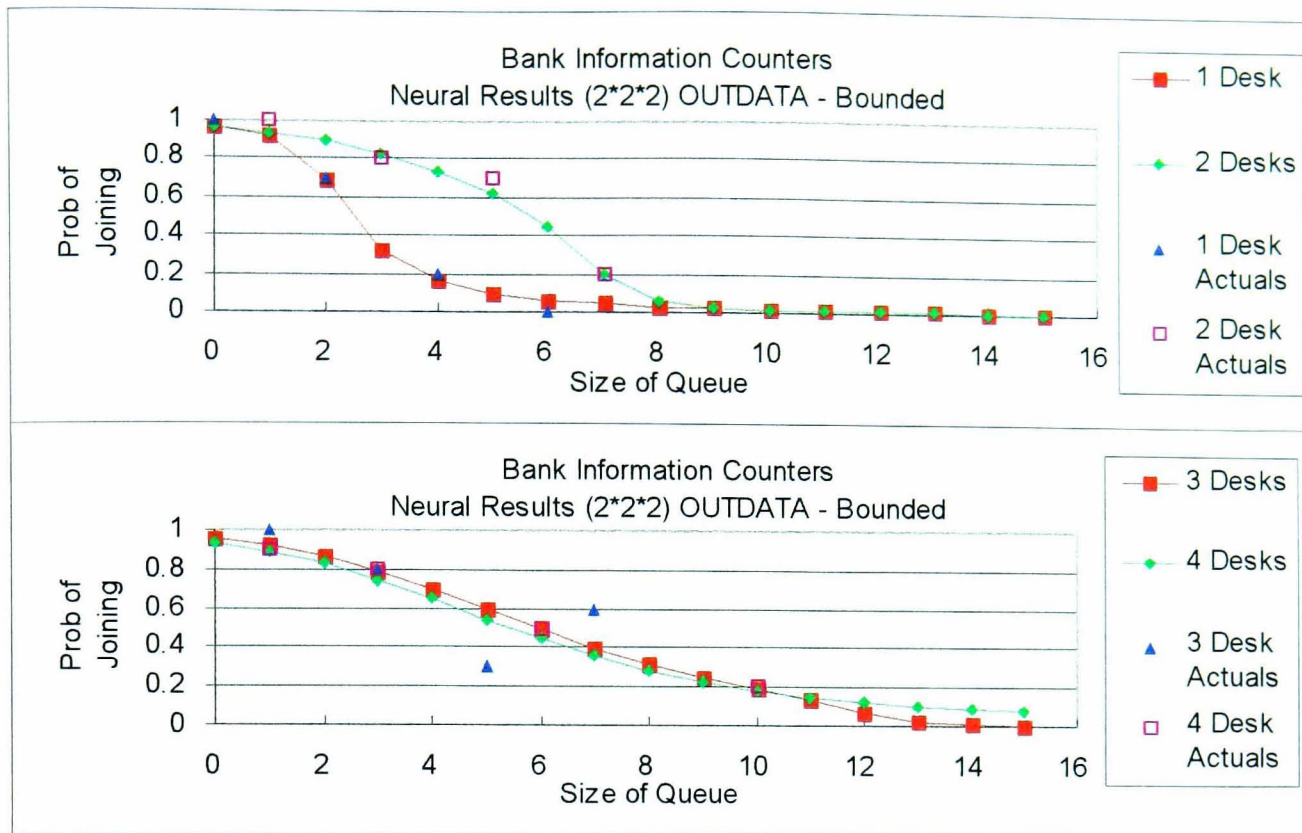


Figure 6.10 : Information Test Results for the Bounded OUTDATA Network

A refinement of the model was tried, by adding artificial data to the data set to improve the tail of the distribution for 4 desks. Two extra data points were added: 3 desks, queue of 13, and 4 desks, queue of 14, both with a zero probability of staying in the bank. Figure 6.10 shows graphs of the resulting model. It can be seen that adding the lower bounds to the data improves the tail of the 4 desk case slightly, but has little other effect.

Allowing the Network to Determine Membership Probability Outputs (OUTNET)

The data set for the OUTNET network approach required least preparation out of all the approaches. It did not require a random number variable or any special preparation for the output targets. As with the other approaches, networks with different numbers of hidden nodes were tried, and were tested using the same data set as that used for the OUTDATA models (the inputs are the same, and the targets are not used).

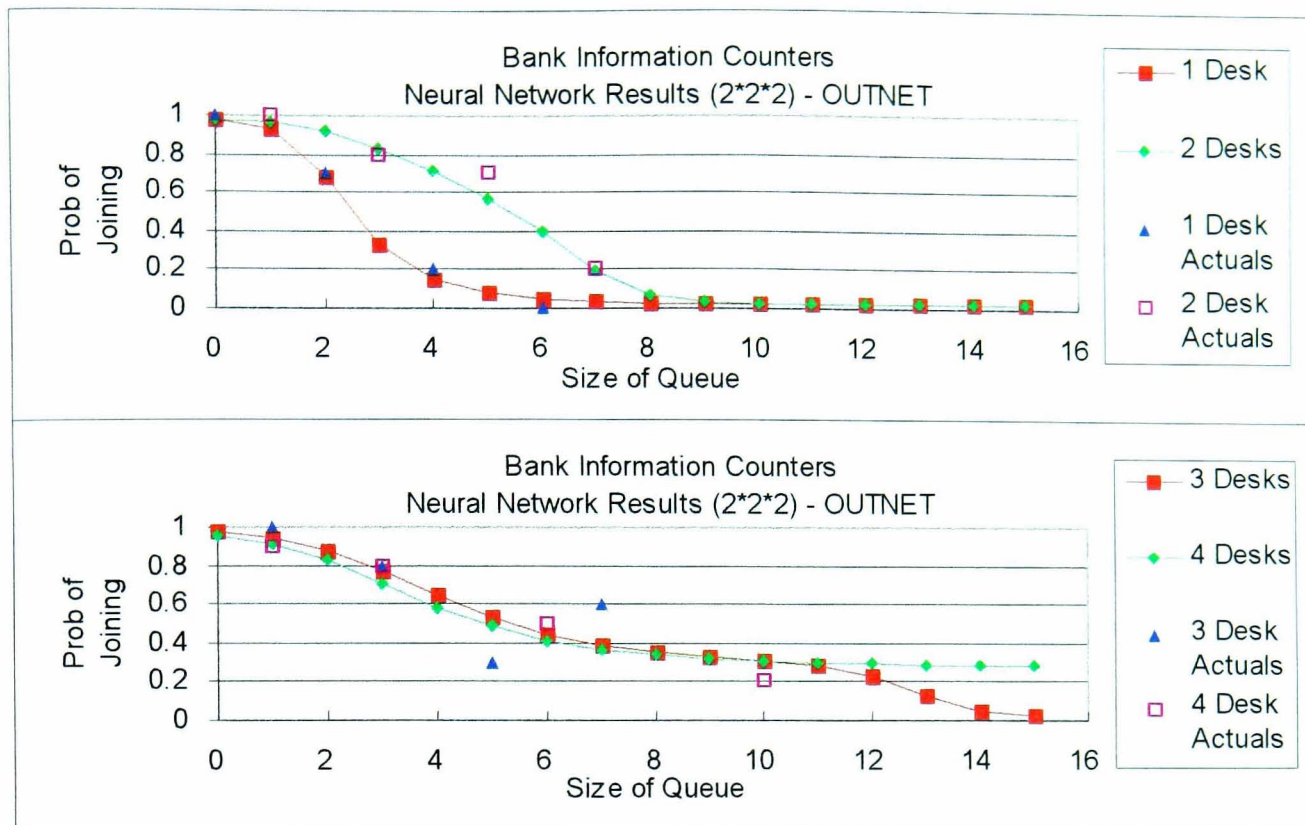


Figure 6.11 : Information Test Results for the OUTNET Network

Figure 6.11 shows the selected network for the OUTNET approach. The results for the other network sizes are shown in Appendix H1. The results obtained for the network sizes show a similar profile to the OUTDATA results.

Artificial boundaries were added to the data set for 3 desks, queue of 13, and 4 desks, queue of 14 both with a probability of staying in the bank of 0. Each of these were added as 10 data points to give them the same weighting as the other data. The resulting network gave results with a very similar profile to the OUTDATA network shown in Figure 6.9.

The Choice of Model

The most internally consistent model is that produced by the INVAR approach. The model most closely follows the expected features of the decision making process. The closed tails of the model compared to the other approaches is a disadvantage but not a serious one, since their effect on the simulation is likely to be negligible.

The structure of the INVAR model is significantly different to the other approaches. To confirm that the model does not represent a 'poor' global minimum, the network

was trained again with different random starting weights, and very similar results to the first INVAR model were achieved.

The decision model needs some rules in addition to the neural network. These are simple cases that the neural network does not cover, which are:

- if all the information desks are closed then leave the bank
- if there are any desks free go to the nearest free one.

The first rule is fairly straight forward since if none of the counters are open then all the customers balk. It is possible in practice that some of the customers may go to different services to try to get information, but even so there would be some customer dissatisfaction. In the case of free desks, the most likely case is that customers would go to the nearest desk (the left-most), but there may be other factors such as a customer having dealt with a particular member of staff before, or that some staff may look more approachable. This is difficult to capture in a simulation, but live observation might help to develop a distribution.

6.5.5.2 Transactions : Join or Balk

The Transactions service uses a multi-server multi-queue system. The first decision is whether to leave the bank or to stay and use the service, with a further decision required afterwards about which queue to join.

The data collection stage had 21 scenarios involving the Transactions service. The data can be generalised better for the join/balk decision if the length of the shortest queue is considered rather than the lengths of all the queues. In a system with a small number of counters the queuing decisions should be reasonably efficient so that the other queues should generally be the same length, or contain one person extra. In large systems such as a supermarket, where customers are unlikely to be able to perceive the states of all the queues, the assumption of efficient queuing is unlikely to be true.

Converting the data to the shortest queue results in scenarios being duplicated in two cases, reducing the number of different cases to 19. Table 6.5 shows a summary of the set of different scenarios for the Transactions service.

Counters Open	Size of Shortest Queue	% Joining
1	1	100
1	3	70
1	5	50
1	7	40
1	9	40
2	2	100
2	3	80
2	5	40
2	7	20
3	2	100
3	4	60
3	7	30
4	2	100
4	3	70
4	5	60
4	7	20
5	2	80
5	4	80
5	6	40

Table 6.5 : Transaction Data showing Percentage of Customers Staying

Note that in the case of Transactions, the queue length includes the person who is being served, so (unlike Information) a queue length of 0 would mean that the counter was free. Looking at the summary results in Table 6.5 shows that in hindsight some longer queue sizes would have been useful to find the points at which very few customers would stay, and it would have been desirable to avoid the replications of shortest queue cases which has slightly reduced the amount of evidence available. When the scenarios were put together there was a concentration on total numbers of customers rather than the lengths of the shortest queues. In many applications it is likely that the data collected will have little evidence for some of the more extreme situations, so that valuable lessons about coping with this sort of data can be gained here.

Another point to note from the scenarios is that in the 5 counter scenarios there are situations where the shortest queues are 2 and 6, however in setting up the scenarios there should have been a total of 16 and 37 people respectively queuing for the service. This suggests that the shortest queues should have been 3 and 7 respectively. The reason for this lies in the scheduling of arrivals. All the queuing customers are scheduled to arrive at the same time so as to build the queues. The human controlled customer is scheduled to arrive 0.1 minute later. The time delay is to ensure that the queues have been properly created before the human controlled customer arrives, however during that 0.1 minute it is possible that an extremely small service time could be sampled and for a customer to have left the bank. This is what has happened in those two cases. The situation does not invalidate the results in any way, but it does mean that those scenarios are slightly different from what was intended. In retrospect a time delay of say 0.0001 minute could have been used which would have prevented the problem.

An analysis of the questionnaires suggests that the length of the queues had the most impact on the decision, with the number of counters being much less important than for the Information service. Some subjects did note that the number of counters had an effect on marginal decisions since more counters would provide better opportunities for queue swapping. Some subjects indicated that they were generally quite tolerant of Transaction queue sizes since they thought they moved quickly, while others noted that the times were highly variable and so were wary of longer queues.

The networks were trained using two parameters, the number of open counters and the length of the shortest queue. Both were scaled down to the range 0 to 1 from the possible range of counters (1 to 5), and from the likely range of queue lengths (0 to 15). Note that scaling the queue length from 0 is used for arithmetic convenience and consistency with the Information model. In fact the neural network model is only concerned with queues of 1 or more.

As with the Information counters, all three approaches discussed in Section 4.4 were investigated, and the results examined using a test set with all possible numbers of

counters (1..5) and a range of queue sizes (1..15). The main observations are summarised below, with the selected model being discussed in more detail. The full set of results can be seen in Appendix H2.

Using a Probability Variable as an Input (INVAR)

The best model was achieved using a 3*4*2 network. The main body of the distribution was generally a sensible shape, with a smooth fit to the data and the right relationships between the number of counters. At the extremes, the tail for the 1 counter case did not tend towards 0 very quickly and for 4, 5 counter cases not all customers joined when there was a queue of 1, despite this being the case for 1, 2, 3 counters.

The models generally suffered at the tails of the distribution by not tending towards 0 for all the possible number of open counters. In the case of the network with 6 hidden nodes, a number of the tails actually started to increase.

To try to improve the model an artificial data case was added which showed that no customers joined the queue when only 1 counter was open and the queue had 10 people in. The 3*2*2 network was much improved having smooth profiles in the main bodies and the tails for all the cases of numbers of open counters.

Representing Class Membership as Outputs in the Training Data (OUTDATA)

The network which produced the results with the best shape had a 2*4*2 architecture. The main bodies were fairly consistent and had a smooth shape, although the 1 counter case had one point where the probability of joining increased with the queue size and it suggested that more people were likely to stay if 1 rather than two counters were open. The tails to the distribution were smooth and tended towards 0.

The larger network with 6 hidden nodes was more seriously affected by increasing proportions of people staying as the queues increased for several of the counter cases. The smaller network with 2 hidden nodes had tails that did not converge to 0.

Allowing the Network to Determine Membership Probability Outputs (OUTNET)

The model with the results having the best shape was from a 2*6*2 network. The main part of the distribution and the tails had a fairly consistent shape for all the counters, except that the model suggested that more people were likely to stay if 1 rather than 2 counters were open once there were more than 4 people in the queue.

The larger network with 8 hidden nodes dealt particularly poorly for the 1 counter case, having an increasing tail that reached high probabilities for a queue with 15 people. The smaller networks with 4 and 2 hidden nodes had poor tails that did not converge towards 0.

The Choice of Model

The OUTNET model and the bounded INVAR model both had reasonably good shapes, being mostly consistent with the expected effects of the decision parameters. A further comparison of the models was made to decide which to implement in the simulation model. Graphs of the test results for both models are shown in Figure 6.12 and 6.13.

Figure 6.12 shows the results for the OUTNET model. The shape of the 1 counter line is rather different to that of the other counters, and one would expect fewer people to stay with 1 rather than 2 counters for the same queue lengths. However, the effect of the training data is very strong on this, with a consistent pattern, rather than just one point being an aberration. The rest of the counters follow a fairly similar pattern to each other with tails that converge towards but do not reach 0. On the whole, the model fits the data quite closely, but exhibits sensible tails for the points where no data exists.

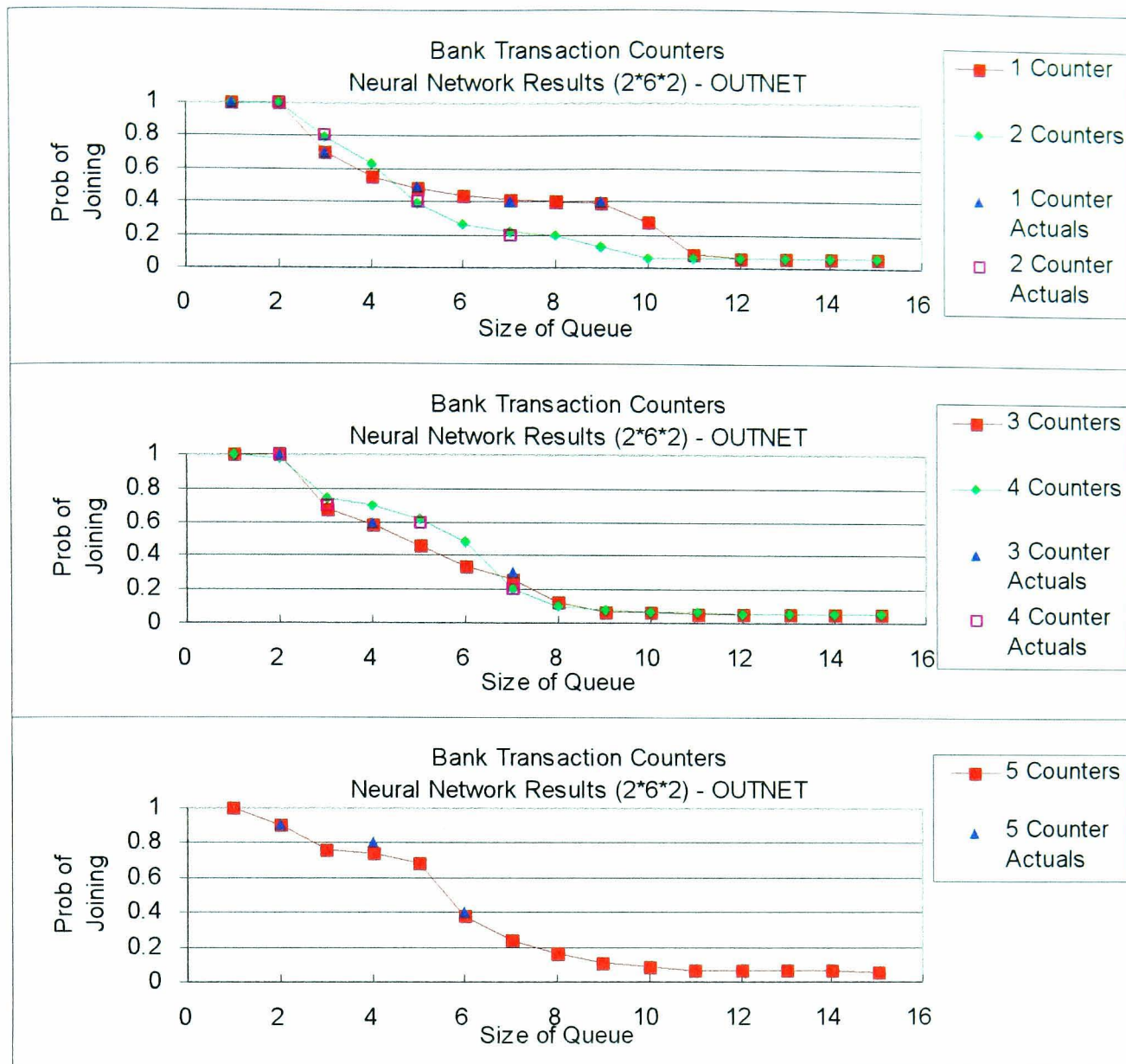


Figure 6.12 : Transaction Test Results for the OUTNET Network

The results for the bounded INVAR model are shown in Figure 6.13. This model has a smoother fit for the main body of the distribution than the OUTNET model. The 1 and 2 counter cases exhibit less difference which matches what would be expected, although there is still a slightly greater chance of staying for the 1 rather than 2 counter case for the longer queues. Similar relationships can be seen between the other counters. This does make some sense, since with several counters open and large queues, the bank area can be seen to be very crowded, which will put some people off more than the shortest queue consideration. The INVAR model is a much smoother fit than the OUTDATA which generates greater confidence in the model, since this is more likely to be closer to the underlying structure. The tails of the distribution are not as smooth as for the OUTNET model, and are closed, each reaching 0 values.

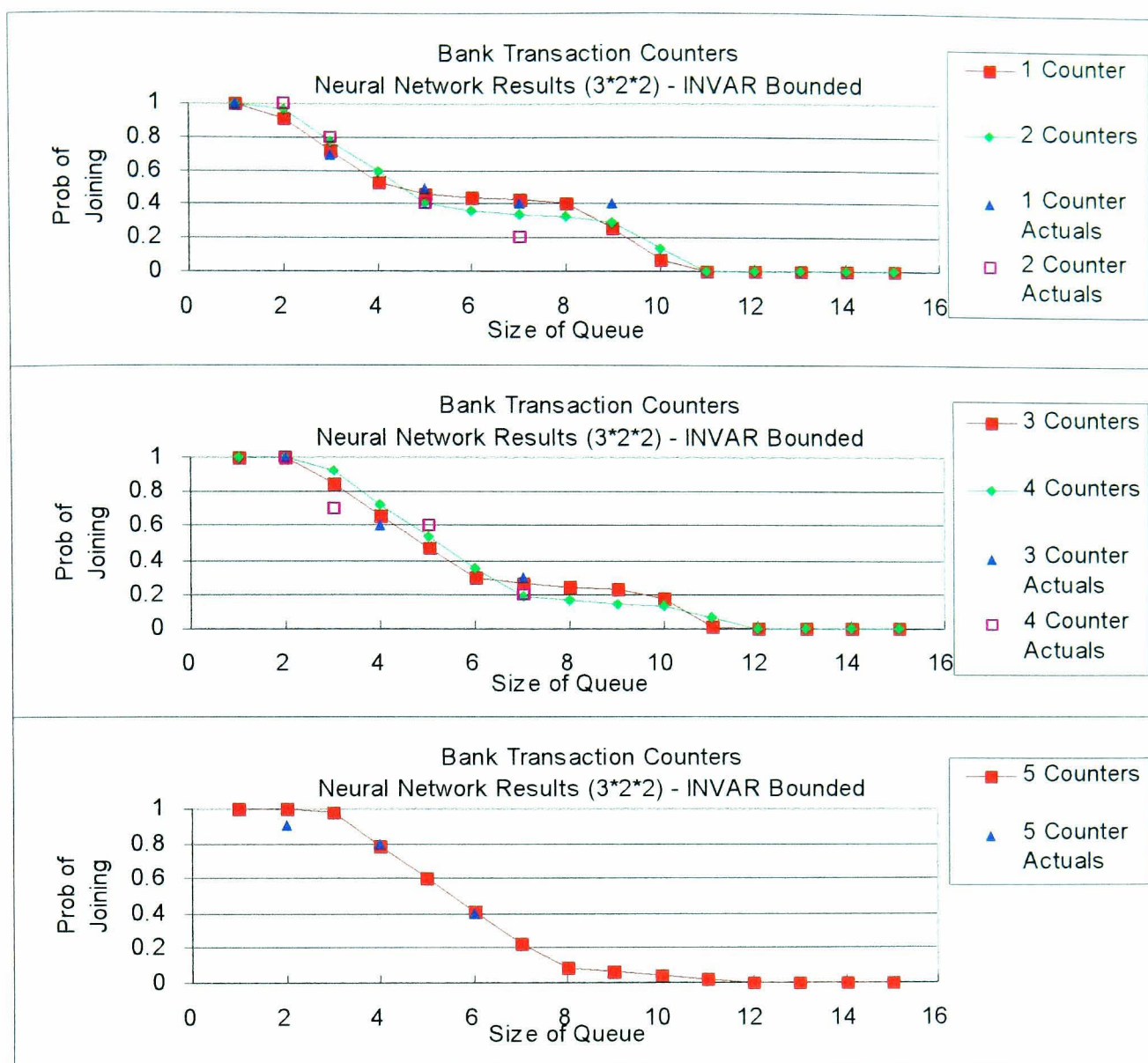


Figure 6.13 : Transactions Test Results for the Bounded INVAR Network

In deciding between the two models, the smoother fit of the bounded INVAR model for the main body, which is more likely to be sampled from in the simulation, overrides the smoother fit of the tails for the OUTNET model. The need for an artificial data case for the INVAR model is a disadvantage, but understandable given the lack of available data for that part of the distribution, and it did have the desired effect of closing the tails.

Simple rules are required to deal with the cases not covered by the neural network, these are:

- if all the Transaction counters are closed then leave the bank
- if any of the Transaction counters are free then stay in the bank

6.5.5.3 Business : Join or Balk

The Business service uses a multi-server, multi-queue system much like the Transactions service, but with a maximum of 2 counters open it is on a smaller scale. It requires a further sub-decision after deciding to join the service to determine which queue to join.

The two decision parameters are the number of counters open (1 or 2) and the length of the shortest queue (scaled between 0 and 15). With only two possible counters, the lengths of the queues for both counters could be represented with the amount of data available. However the use of the shortest queue makes better use of the data available, since there is less need for interpolation with little loss in the accuracy of the model.

The data collection stage included 10 scenarios involving the Business service. In one case, two scenarios have the same shortest queue length, leaving 9 distinct cases. Table 6.6 shows a summary of the data collection results for the Business service.

Counters Open	Size of Shortest Queue	% Joining
1	1	90
1	2	70
1	4	30
1	8	10
2	1	95
2	2	90
2	4	50
2	6	20
2	8	0

Table 6.6 : Business Data showing Percentage of Customers Staying

The data looks fairly well behaved with a monotonic reduction in the chance of staying as the queue sizes increase. On the whole customers are more likely to stay if there are two counters open, though the exception to this is the case of 8 people in the queue. Intuitively this last case seems to be an aberration, and it is hoped that the networks would cope with this.

The results of the models from using the test data are shown in Appendix H3. The main outcomes are summarised below.

The questionnaire suggested that most people treated the Business service as a small version of the Transactions, concentrating mainly on the shortest queue length, with some consideration of the number of open counters in marginal cases. Interestingly, some people indicated the Business service as the quickest for being served, while others suggested it was the slowest.

Using a Probability Variable as an Input (INVAR)

The model with the best features used a 3*3*2 architecture. The model follows the data quite closely, interpolating smoothly between them. The data suggests that more people will stay for one rather than two counters open when the queues are large. This is counter intuitive, but is represented in the model, though there is only a slight difference between the 1 and 2 counter cases at the tails. The tails are closed, with no people staying for a queue of 9 or more.

The networks with more hidden nodes fit the data more closely, particularly at the tail (which is not a good feature) and the interpolation between points is not as smooth. The smaller network fits the model reasonably well at the top end, but has very short tails.

Representing Class Membership as Outputs in the Training Data (OUTDATA)

The model with the best shape uses a 2*4*2 architecture. The model fits the training data very closely, but does smooth interpolation between the points. The tail for 2 counters is reduced to very small probabilities for a queue size of 8, but dies away only slowly for 1 counter.

The larger network with 6 hidden nodes has a similar main body, but the tail for the 1 counter does not tail off towards 0. The smaller networks have a similar shape to the best network, but tail off lightly more slowly.

Allowing the Network to Determine Membership Probability Outputs (OUTNET)

The best model has a $2 \times 4 \times 2$ architecture. The model fits closely to the training data but does not interpolate smoothly between them. The tails are smoother than the main body, though again the probability of staying with 1 counter open tails away more slowly than for two counters.

The model with 6 hidden nodes interpolates slightly more smoothly between the data points, but has very short tails. The smaller networks are still jagged, and do not tail away quite as well.

The Choice of Model

The OUTNET model was not a smooth fit, which does not lead to confidence in the model. The OUTDATA and INVAR models had similar shapes and smooth fits. The INVAR model did not exaggerate the difference between the tails of the 1 counter and 2 counter cases as much as the OUTDATA model, and so is the model of choice. Figure 6.14 shows the test results for the INVAR model.

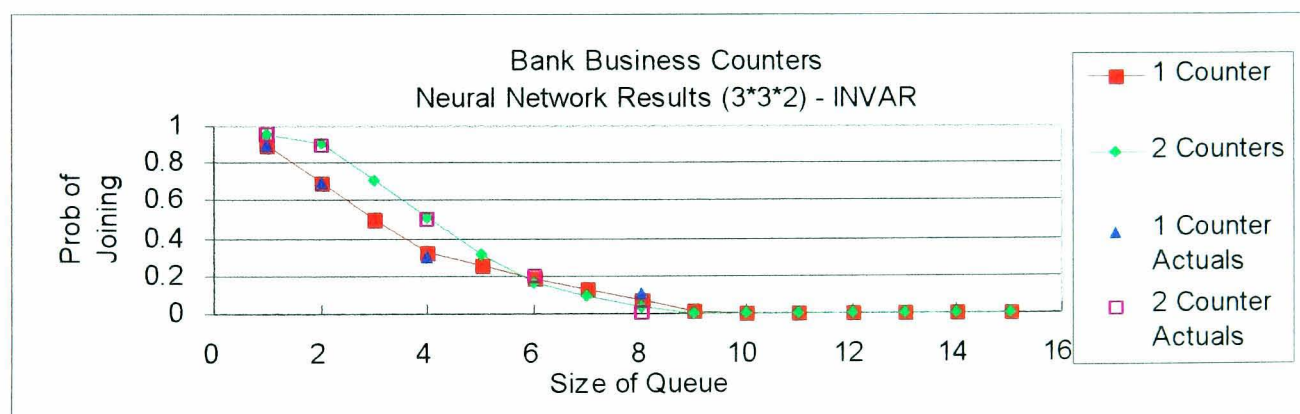


Figure 6.14 : Business Tests Results for the INVAR Network

Simple rules are required to cover the cases not represented by the network, these are:

- If all Business counters are closed then leave the bank
- If any of the Business counters are free then stay in the bank

Business services are one case where the customer does have the option of going to the Transactions counter for some services or the Information counter for others. A probability distribution could be used to determine which service the customer looks at

instead of Business, or whether they just leave. The balk profiles for the Business customers who use the other services may well be different to those for the ordinary customers. This might be consideration for a larger study, but will be ignored for the purposes of this investigation.

6.5.5.4 Currency : Join or Balk

The Currency service queuing system works in the same way as that for the Business service. The decision criteria are the number of counters open (1 or 2) and the size of the shortest queue (scaled between 0 and 15). The data collection stage involved 10 scenarios for the Currency service, the results of which are summarised in Table 6.7.

The data shows a number of difficulties for fitting a smooth neural network model. There are several points where the percentage staying in the bank stays the same as the queue increases, and at the end of the 2 counter case, the percentage staying increases. An understanding of the decision process suggests that the decision profile should be monotonic, there could be flat spots, but the percentages staying should not increase with queue sizes.

Counters Open	Size of Shortest Queue	% Joining
1	2	80
1	3	80
1	4	40
1	6	20
2	1	90
2	2	70
2	3	50
2	4	50
2	5	10
2	6	30

Table 6.7 : Currency Data showing Percentage of Customers Staying

The questionnaires showed that in many cases, subjects treated Currency in the same way as Business. Some people viewed the service as quick, others slow, and one person thought it was the most unpredictable in terms of service times.

The results from the neural networks are given in Appendix H4, and summarised below.

Using a Probability Variable as an Input (INVAR)

The best model used a 3*2*2 architecture. For small queues (up to 3 in the queue) the model suggests that a greater proportion of people are likely to stay with 1 counter open rather than 2. This is counter intuitive, but is strongly represented in the training data. The distribution tails off fairly smoothly, but in a near linear fashion.

The larger networks fail to tail off completely or do so slowly. The network with 6 hidden nodes fits fairly closely for small queues but does not produce a smooth model.

Introducing artificial bounds to the data, suggesting that no people will join the when the queue is 8 long for one or both counters open, produced a slight improvement in the smoothness of fit in the model. The 3*2*2 network has only a small difference between 1 and 2 counter cases for small queue sizes, and both tail to 0% staying at a queue size of 8. The resulting model is close to linear, but is this is a sensible compromise given the inconsistent nature of the training data.

Representing Class Membership as Outputs in the Training Data (OUTDATA)

The network which gave the smoothest fit had a 2*3*2 architecture. It fitted quite closely to the data, but did treat the inconsistent data for 2 counters sensibly (taking the average of the two). However, the network failed to tail off, giving a minimum percentage staying of just over 20%.

The larger networks produced a poor general model which fitted the data closely, but then had larger percentage of customers staying as the queue size increased, resulting in a U-shaped model. The smaller network gave similar results to the best case.

Using artificial data to bound the training data in the same way as for the INVAR approach results in a 2*2*2 network with a smoother fit and tails that tend towards 0.

For smaller queue sizes there is a sizeable difference of up to 20% more people staying for the 1 counter case over the 2.

Allowing the Network to Determine Membership Probability Outputs (OUTNET)

The network with the smoothest fit had a $2 \times 2 \times 2$ architecture, giving very similar results to the best case for the OUTDATA network.

The larger networks fitted the data very closely, but had poor tails that increased rather than converging down to 0.

Using artificial data to add a tail boundary, as for the other methods, the resulting best model has a $2 \times 3 \times 2$ architecture. This does have tails that smoothly converge to 0, but does not have a smooth fit for the smaller queue sizes, with nearly 30% more people staying if 1 counter is open rather than two where there are 3 people already in the queue.

The Choice of Model

Of the models fitted using the original data, the clear choice is the INVAR model since this is the only one that tails off to 0% staying in the bank. However, comparing the fit for the 1 and 2 counter cases shows that they do not follow the expected general relationship between the two. The results from the model can be seen in Figure 6.15.

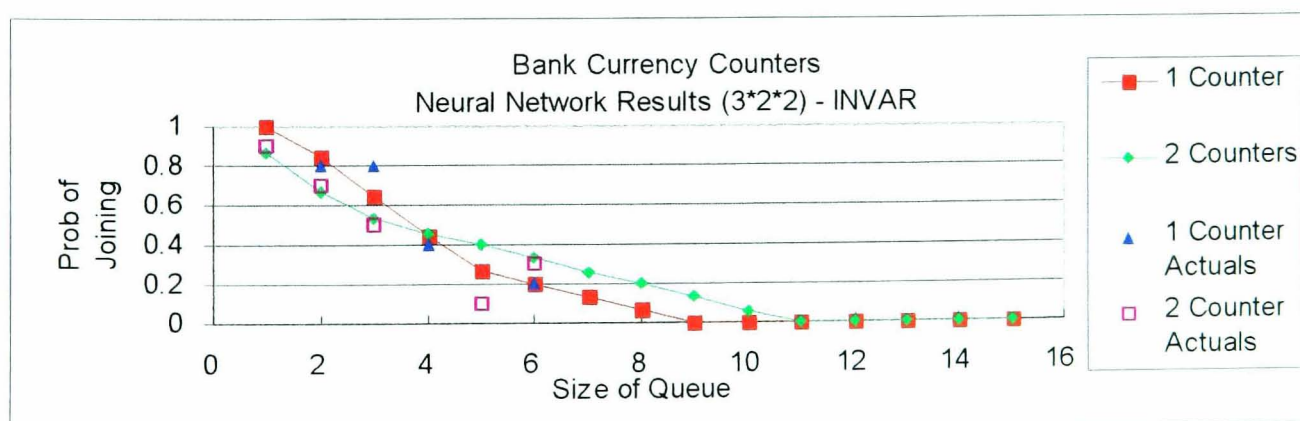


Figure 6.15 : Currency Results for the INVAR Network

For the networks with the artificial data added, both the INVAR and OUTDATA approaches produce models with smooth fits. These are shown in Figure 6.16 and Figure 6.17.

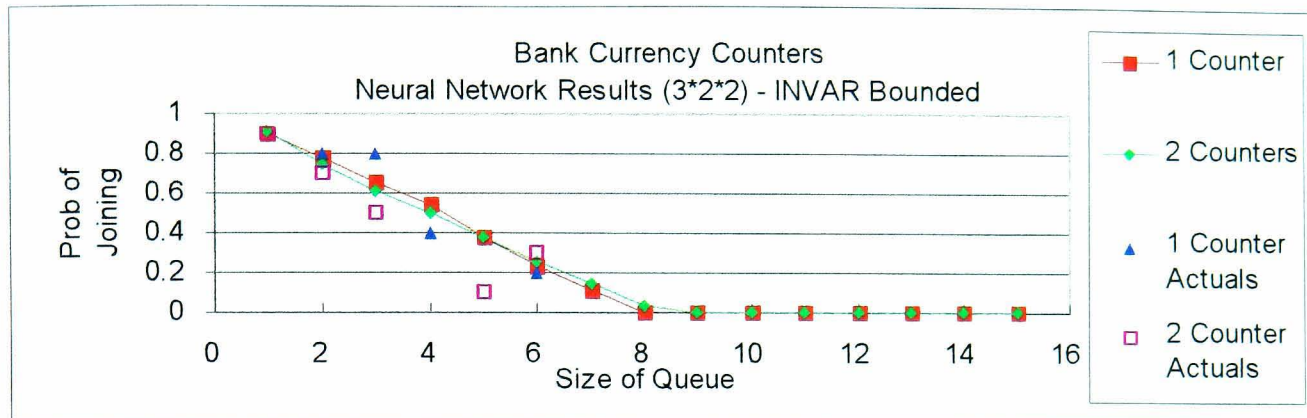


Figure 6.16 : Currency Results for the Bounded INVAR Network

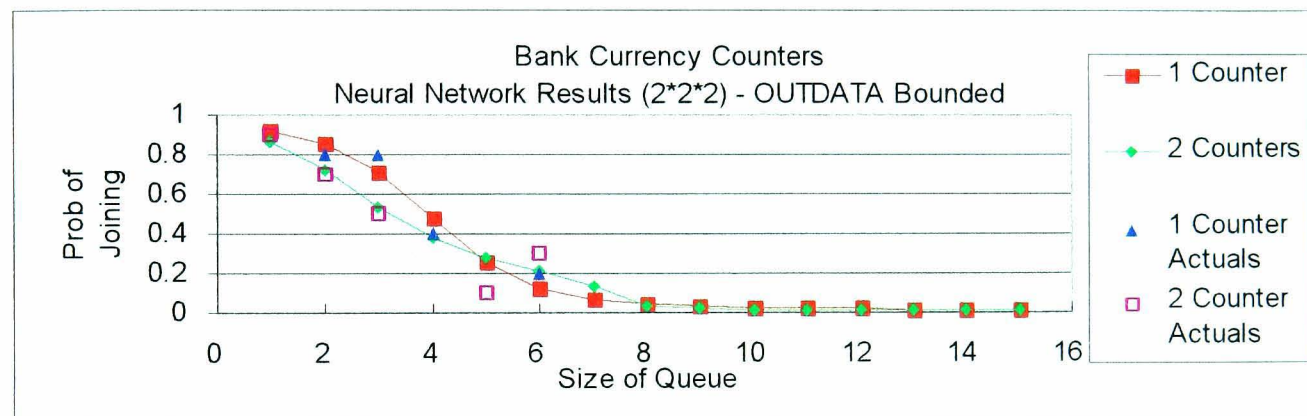


Figure 6.17 : Currency Results for the Bounded OUTDATA Network

The Bounded OUTDATA model has smooth tails and reasonably smooth overall shapes for the 1 and 2 counter cases taken separately, but does not follow a relationship between the two that would be expected from a general understanding of the decision process. The more linear shape of the Bounded INVAR model is not as smooth, but has a much smaller difference between the counters. In view of this, the Bounded INVAR model is the preferred one.

As for the previous services, the extra rules account for the situations not covered by the network model, which are:

- If all Currency counters are closed then leave the bank
- If any of the Currency counters are free then stay in the bank

Another possible option for the Currency service is to consider it as the same decision model as the Business service and combine the data. Comparing Tables 6.6 and 6.7 suggests that such an argument could be made. However, the model developed for Currency here is usable, and little further benefit for evaluating the stochastic neural network approach would be gained by developing a further model.

6.5.6 Queue Joining Decisions

If a customer has decided to stay in the bank for their desired service, the next sub-decision involves the queue that they should join. Information only has a single queue, so no decision is needed. The other service types will be considered here.

The data collection framework may introduce some bias into the choice of queues. When the subject is presented with a choice of queues, the first available queue is highlighted by default. For those people who are controlling the simulation themselves, the option with least keypresses is to choose that default queue. Whilst this might not be ideal for collecting data about the real situation, for analysis purposes here it represents a valid preference.

6.5.6.1 Transaction Queue Decision

The Transactions service can have between 1 and 5 counters open. The case of 1 counter open is a trivial one, since there is no choice, so analysis will only be done for more than 1 counter.

Table 6.8 shows a summary of the counters and queue sizes, and the number of subjects who chose to join each queue. Each column in the table shows which queues were open, the first number indicating how many customers were in the queue, and the second number indicating how many of the test subjects (who did not balk) decided to join that queue. The cases where subjects chose a queue that was not the shortest are highlighted.

Counters Open	Queue 1 size/no. joining	Queue 2 size/no. joining	Queue 3 size/no. joining	Queue 4 size/no. joining	Queue 5 size/no. joining
2	-	-	-	2 / 10	2 / 0
	-	4 / 2	-	3 / 7	-
	-	5 / 2	5 / 2	-	-
	-	-	-	7 / 2	7 / 0
3	-	2 / 6	-	2 / 3	2 / 1
	3 / 1	3 / 0	-	2 / 9	-
	-	5 / 0	4 / 5	-	4 / 1
	7 / 1	-	7 / 1	-	7 / 1
4	3 / 4	-	3 / 2	3 / 0	3 / 1
	3 / 1	2 / 3	-	2 / 2	2 / 4
	5 / 1	5 / 1	5 / 2	5 / 2	-
	7 / 2	8 / 0	-	8 / 0	7 / 0
5	2 / 5	2 / 0	2 / 3	2 / 1	2 / 0
	3 / 2	3 / 0	3 / 0	4 / 0	2 / 5
	5 / 0	4 / 7	5 / 0	5 / 0	5 / 1
	6 / 4	8 / 0	7 / 0	8 / 0	7 / 0

Table 6.8 : Transaction Queue Choices

Looking at the cases where subjects did not join the shortest queue, in most of them the queue joined was the default one (the queue closest to the top), and may indicate that they pressed the wrong button on the keyboard.

A Neural Network Approach

The most desirable neural network model would be one that had 5 inputs, each representing the size of the queue for each counter or that the counter was closed, and 5 outputs each giving the probability of joining that queue. This sort of network would be very specific since it would need to cope with a variety of patterns of open and closed counters (46 different patterns are possible). For each pattern, there would be a large number of different queue sizes to be coped with (theoretically infinite, but using likely values there are still many thousands).

To reduce the dimensionality to some degree, there could be concentration on only three possibilities for each queue : shortest, not shortest, or closed. This gives 196^* different combinations that the model would need to deal with. Even if the neural network was trained with only 10% of the examples, this would still require 20

* $2^5 - 1 + 5 \cdot (2^4 - 1) + 10 \cdot (2^3 - 1) + 10 \cdot (2^2 - 1) = 196$ (note: cases with no shortest queues are ignored)

scenarios for which data would be required. With the spatial nature of the model, the structure of the network makes it a difficult task to interpolate adequately from the training data because the patterns are highly specific. Given the constraints on session times and number of test subjects available, this would not be practical for the Bank simulation model.

Since it is the spatial relationship that is of interest, the situation cannot be generalised any further to allow the neural network to model the situation with a limited amount of data.

An Empirical Probability Approach

While the main interest of the study is to look at plausibility of using the stochastic neural network approach for representing the decision making, for completeness, an alternative to a simple random allocation of customers to queues is suggested.

Probabilities can be calculated for the popularity of each counter from the data, taking account of the biases introduced by the observed patterns of counters. For instance, looking at Table 6.8 it can be seen that there is an instance for Queue 4 (3 counter case, 2nd row) where that is the only counter with the shortest queue, and consequently 9 out of 9 people were seen to choose that queue. On the other hand, Queue 3 has no situations where it is the only shortest queue. Thus the results would be biased in favour of Queue 4. Also, different scenarios had different balking rates observed, so that the total number of people joining the queues differs in each case.

The analysis involved only looking at the shortest queues (there could be several of the same length), and for each of these queues the observed number of people joining was recorded. Then the expected number of people who would join the queue, given that each is equally popular was calculated. Again this only considers a queue when it is one of the shortest. For instance, in the first row of data in Table 6.8, Queue 4 and Queue 5 are open, both with two people in the queue. Since they both have the shortest queue, they are both considered. In total 10 people joined a queue, so the

expected number of people would be 5 in each queue (in fact all of them preferred Queue 4).

A weighting factor of Observed/Expected can be calculated in each queue, and used to calculate the probability of joining the queue given any combination of shortest queues. The results of the analysis for the Transactions queuing data are shown below.

	Queue 1	Queue 2	Queue 3	Queue 4	Queue 5
Observed (O)	17	19	15	36	13
Expected (E)	10.2	18.6	11.7	33.8	25.9
Weighting W=(O/E)	1.667	1.022	1.282	1.065	0.502

If, say, Queues 1, 3 and 5 are the shortest, then the probability P_1 of joining Queue 1 is:

$$P_1 = \frac{W_1}{W_1 + W_2 + W_3} = \frac{1.667}{1.667 + 1.282 + 0.502} = 0.483$$

The approach measures the popularity of each counter relative to the others, but does not take into account the interaction effects of particular patterns of counters. The cases of subjects choosing a queue that is not the shortest will be ignored.

6.5.6.2 Business Queue Decision

The Business service involves a maximum of only 2 queues, so the queuing decision is on a much smaller scale than that for Transactions. There is no choice where only 1 counter is open, so that decision is trivial. The 2 counter open decision will be looked at further. The queue choices from the data collection are shown in Table 6.9. Note that there are no cases of someone joining the longer queue.

Queue 1 size/n° joining	Queue 2 size/n° joining
1 / 10	1 / 0
2 / 0	1 / 9
2 / 4	2 / 5
5 / 0	4 / 5
6 / 1	6 / 1
8 / 0	8 / 0

Table 6.9 : Business Queue Choices

If like Transactions only the cases for shortest (S), and not shortest (N) are considered (closed counters are not of interest here), then there are only options S N, S S, N S to consider. The dimensionality of the decision is not an issue as it was for Transactions. Table 6.10 shows an analysis of the results for the reduced problem.

Situation	N° People Joining (1st Queue / 2nd Queue)	Total Joining (1st Queue / 2nd Queue)	Probability of Joining (1st Queue / 2nd Queue)
S N	-	-	1 / 0*
S S	10 / 0, 4 / 5, 1 / 1	15 / 6	0.714 / 0.286
N S	0 / 9, 0 / 5	0 / 14	0 / 1

* From symmetry with the N S case

Table 6.10 : Business Queue Choices for the Reduced Problem

The scenarios provide evidence for two out of the three cases. Tests with the neural network showed that some interpolation for the S N case was possible. The best case was with the OUTDATA model, which modelled the N S and S S cases exactly but for the S N case it only produced a probability of 0.83 for people joining the shorter queue, whereas a value closer to 1 would have been expected. The INVAR model did very poor interpolation giving the same probabilities for S N as it did for S S.

The best estimate for the N S case is to treat it as being symmetrical with S N, as shown in Table 6.10. Since the neural network does not help in this decision, the probabilities in Table 6.10 will be used.

6.5.6.3 Currency Queue Decision

The queue decision for currency is much the same as that for Business. The results from the scenarios are shown in Table 6.11. The data contains one case of a subject joining the longer queue.

Queue 1 size/n° joining	Queue 2 size/n° joining
2 / 1	1 / 8
2 / 7	3 / 0
3 / 4	3 / 1
4 / 4	4 / 1
5 / 0	5 / 1
6 / 1	6 / 2

Table 6.11 : Currency Queue Choices

Reducing the situations down to shortest (S) and non-shortest (N) queues, gives the observations shown in Table 6.12. The one case of a subject joining the longer queue is likely to be a mistake through pressing the wrong key on the computer keyboard. The probability shown for the N S case of joining the longer queue is very likely to be over-emphasised, but it will serve for demonstration purposes.

Situation	N° People Joining (1st Queue / 2nd Queue)	Total Joining (1st Queue / 2nd Queue)	Probability of Joining (1st Queue / 2nd Queue)
S N	7 / 0	7 / 0	1 / 0
S S	4 / 1 , 4 / 1 , 1 / 2	9 / 5	0.643 / 0.357
N S	1 / 8	1 / 8	0.111 / 0.889

Table 6.12 : Currency Queue Choices for the Reduced Problem

A neural network with enough hidden layers is able to work as a look-up table. Given that a complete set of data is available for the Currency counters, a neural network was able to represent the data completely, even with no hidden layers. In this case, the ability to use the network as a look-up table is no real benefit, since the original probabilities can easily be used on their own. In the simulation, the probabilities for the two shortest queues case will be used, otherwise the customer will join the single shortest queue.

6.5.7 Reneging Decisions

Reneging involves customers joining a queue and then deciding to leave the bank without waiting to be served.

6.5.7.1 The Decision Task

Once customers have joined a queue, there is the option to leave the bank (renege). In the bank simulation, subjects were allowed to renege from queues, taking the exit penalty plus the amount of time they had already spent queuing.

In all of the data collection trials only 15 instances of reneging were observed. The details of these cases are presented in Table 6.13.

Service	Number of Counters Open	Shortest Queue Length	Position in Queue	Time in Queue (mins)
Information	2	6	5	2.45
	2	6	5	4.42
	2	7	6	1.10
	3	8	7	1.00
	4	5	4	1.14
Transactions	1	7	5	4.14
	3	4	5	1.10
	3	5	7*	1.50
	5	6	6	4.40
Currency	1	2	1	11.40
	1	3	3	1.55
	1	5	4	1.01
	1	7	5	2.87
	2	3	6*	5.01
	2	4	5	0.25

* Possible mistake by subject

Table 6.13 : Summary of Reneging Cases

In the reneging observations, the Information, Transactions and Currency services are all represented, but not Business. Two observations are suspicious in that the subject could have improved their queue position by changing queues, but instead elected to renege. This could be done by making a mistake in their choice when using the simulation. Table 6.14 shows the proportion of customers for each service who reneged (including the two suspicious cases).

Service	Reneged	Total Initially Joining Queue	% Reneging
Information	5	100	5.0
Transactions	4	139	2.9
Currency	6	53	11.3
Business	0	55	0
Total	15	347	4.3

Table 6.14 : Proportion of Customers Reneging for each Service

In over half of the cases, the subjects had been waiting less than 2 minutes before deciding to renege. This suggests that instead of getting fed up with waiting, the subject had re-evaluated how long they were likely to spend queuing and decided to leave the bank. These cases could be considered as being a delayed balk. The other cases involved waiting for a longer period of time, with one (11.4 minutes) being substantially longer. For most of these longer cases, the customer was still a long way

back in the queue, so there was little hope of getting served soon. In the case of the longer wait, the customer was almost in the position to be served.

This suggests that there are two forms of reneging, the re-evaluation of the balking decision, and the disenchantment with waiting. However, in the vast majority of cases the customers stayed until served. An analysis of the questionnaires backs up this view. Some of the subjects stated that they were constantly re-evaluating their decision to stay in the bank in light of the progress in the queue. Others stated that they considered leaving the queue if it hadn't moved for a while, i.e. they were frustrated by inactivity in the queue. However, the majority of subjects said that either they had faith in their original decisions and stuck to it, or they felt that they had invested time in the queue and so were unwilling to leave it.

For each customer, the queuing data shows a series of snapshots for when they were in the queue. Each time the state of the queue changed (i.e. the queue shortened, or the customer decided to change queue or leave), the state of the system and the time in the queue was recorded. Looking at the sequences for each customer who reneged, shows that in all but one case the queue had moved no more than twice. This demonstrates that it is not so much the total length of time spent queuing that has an effect of the decision, but the rate of change in the queue length. When customers first make the decision to stay, they accept the length of queue and have an expected queuing time. If however the queue is moving more slowly than they thought it would, then they re-evaluate their expected times and decide whether to leave or not.

6.5.7.2 Issues for Building a Neural Network Model

The data set is large, but within that the number of examples of reneging is small. This makes it difficult to adequately capture the criteria which make reneging more likely. The limitations in the data mean that the number of dimensions, and therefore the decision criteria that can be modelled is limited. It is still useful to attempt to build a model because this decision involves features not covered by the balking decision, and so adds to the debate about the practical implications of using the stochastic neural network approach.

In the balking decisions, there was a definite point in time at which the decision had to be made, i.e. upon arrival at the bank. In the case of reneging, there is no definite point in time. The question is not only if a customer will renege, but also when.

The renege decision has a continuous time-line in that it can happen at any moment from the time that the customer joins the queue until the time that they are served. One option, which is sometimes used in simulation modelling, is to generate a maximum time that a customer is prepared to stay in the queue that is determined when they first join the queue. However, this does not take into account the dynamic issues of the way that the situation changes once the queue has been joined. The other option is to re-evaluate the decision to stay in the queue. This does not fit well into the discrete event framework since either the decision is re-evaluated every time an event occurs, in which case it will not be at regular intervals, or decision points need to be scheduled which puts an extra computational burden on the running of the simulation. Despite these problems, the re-evaluation approach seems to offer a higher degree of decision making model accuracy, and so will be considered further.

In a simulation which uses time-slicing to model a continuous process, such as a motorway simulation, a number of checks and calculations are required during each point in time, so that the overheads of checking a neural network decision model are not high. In the case of the bank simulation, time advances would only usually occur when an event occurs, such as the arrival of a customer, or a customer being served. Scheduling extra events purely for evaluating the reneging decision carries all the overheads of the operations that the simulation engine would normally have to do for an event (such as graphical updates, checking C-phase routines etc.) and so it will have an impact on the operating speed of the simulation. Checking the decision making model only when some other event occurs is generally going to be accurate enough for most purposes. Whichever approach is taken, care needs to be taken so that the frequency with which the decision is made does not bias the results.

If a random number were to be sampled each time the decision was made, then the more that the decision was tested, the more opportunities there would be to generate a value that caused a reneging event. There is also the issue of consistency. Generating a variety of random numbers for an individual customer could be regarded as allowing quite different attitude swings for that customer. This is clearly brought out when considering a car driver deciding if they want to overtake another car. One moment a random number might suggest that they have a very cautious attitude to overtaking, the next random number might have them acting impetuously.

A solution is to generate a random number for a customer, for a particular decision, and stick to it for all further re-evaluations of that decision. Thus a change in the decision would be down to a change in the circumstances rather than a change in the random number. Different customers would still maintain variety in their decision making, but an individual would act more consistently.

A problem in using a fixed random number does occur in the reneging situation. As has been observed, there are likely to be two types of decision process going on, a delayed balk and disenchantment. Treated as one decision process, it is likely to be a bi-modal decision, thus a number of people who may have been disenchanted and left, have already been caught by the delayed balk peak.

The renege decision could be treated as different decision processes with different random numbers being sampled for each process. The different processes indicated by the value of some decision parameter. The other possibility is to treat the process as one, but to make the probabilities cumulative, so that there is an ever increasing chance of reneging, but the rate of increase varies to reflect the bi-modal distribution. Those customers who have already balked have gone, while the increasing probability brings more people into scope for balking.

The reneging data shows a sequence of queue states. A number of parameters that could affect the decision can be identified : service type, number of counters open, position in queue, time in the queue, progress made in the queue. With the extremely

limited number of renege cases, it is not possible to include all of these decision criteria.

A number of the questionnaires indicated that people thought about renege when the queue had been stationary for a while. This is not a very stable indicator since a customer who has made rapid progress through the queue, and then has a longer wait is less likely to be discontented. A related measure was looked at which considered the average wait in the queue per customer served. This is more likely to capture discontent about a slow moving queue. This was calculated as the time spent in the queue at any point in time, divided by the number of people who have been served (or divided by 1 if no-one has been served).

The average service time was calculated for the point in time at which customers renege, and for those customers who did not renege the worst average service time at any point during their queuing was calculated (on the basis that this represented the time when they might be most tempted to renege).

Looking through the data, it was also apparent that people were more likely to renege if the queue was longer rather than shorter. Since queue positions ranged from 1 to 10 in the data set, considering each of those separately would overspecify the model given the small number of renege examples. Thus, two classes were created : close to front (3rd or closer in queue) and further from the front (4th or further back in the queue).

Ideally, it would be better to have different decisions for each of the service types, since people's expectations for how fast the queue should move differed for the services. Also, how far the person had moved in the queue since joining is likely to be an important factor, although it is hoped that this will partly be picked up in the average service times. Unfortunately, the data restrictions mean that these cannot be included in the model.

The data set which was formed using the above scheme has two input variables: queue position class (0 = close, 1 = further back) and average service time for customers

while in the queue. The data set contains 348 cases, of which 15 are for reneging customers.

The next issue is to determine the probabilities for reneging. There are several points to consider in doing this. Data points are not evenly spread through the range of possible values, and so some regions of the decision criteria are better represented than others. In particular there are far more observations with short queuing times and positions close to the front of the queue than for further back in the queue and long queuing times. Observations for other areas of the data set have very few examples, for instance an average service time of 11.4 minutes is a definite outlier and was observed for only 1 customer (who reneged).

The idea of having a cumulative probability surface is an attractive one. A random number can be generated, and then changes in the state of the queue can move the customer either nearer or further away from the conditions under which they would renege. From the snapshots of behaviour in the data, coming up with a set of cumulative probabilities for what is effectively a multi-dimensional probability distribution is extremely difficult, particularly since there is no functional form for the probability distribution to work with. There is no obvious method to do this either manually, or to get the neural network to automatically generate the cumulative probabilities from the data set.

A less idealistically attractive approach, but a more practical one is to split the data set down to a set of groups, where the key decision variable is allowed to vary (ideally a time related one), but the others are fixed, and to calculate cumulative probabilities along that one varying dimension for each group of data. Each group of data is treated as an individual decision process, although the groups could be modelled using a single neural network model. A single random number is generated and used while the customer stays within the bounds of the decision group, but a new one is generated when they move into another decision group.

The variable chosen to generate the cumulative probabilities against should be one of the most explanatory variables, and one whose value varies a great deal. In the renege decision this is clearly the average service times, with the decision being split into two groups based on the queue position class. This still leaves the issue of different parts of the decision criteria region having disproportionate amounts of data available. Lots of small average service times with few examples of renegeing will tend to swamp a few larger service times with a greater incidence of renegeing, since they will have the same denominator (the total number of examples in the decision group). This is not a problem with the previous balking decision models and the ones based on theoretical data in Chapter 4 because there all the probabilities were relative only to those examples with the same sets of criteria. Here, trying to generate the cumulative probabilities involves looking across the range of values for a decision criteria that is allowed to vary.

A solution to at least reduce the effects of the inequalities of available examples, is to split the average service time variable into smaller groups which then vary in a more restricted range. This means that the decision criteria values in the group are more homogenous, and so specifying probabilities only in relation to the reduced group gives a better estimate of the true rates for the less frequent conditions. The degree to which this can be done is limited by the need to have sufficient amounts of data in each group to build a reliable model. With only few instances of renegeing, groups that are too small will have regions where the rate of balking is under-emphasised, and others where it is exaggerated, depending on where the boundary between the groups is placed. Larger groups will result in smoother models but will be less homogeneous and so more prone to the swamping effect discussed earlier. A balance must be struck between these two competing considerations.

6.5.7.3 The Network Training Data

The data was prepared by classifying each example according to the queue position (class 0 if 3rd from the front or nearer, class 1 if further than 3rd from the front), and specifying the maximum average service time while queuing (or the average service

time at the point the customer reneged). These were sorted, first by position class and secondly by the average service time, both in ascending order.

In selecting the groupings for the data, there was an immediate split to account for the two queue position classes. In splitting the average service time into groups, there was an awareness that with only 15 reneging examples the groups had to be large so as to generate a fairly smooth model. The task was slightly helped by the fact that there are definite groupings in the data, with large areas of the data set having no reneging. With 338 data points, it was decided that there ought to be 7 groups with approximately 50 data points in each. With similar group sizes, the denominators for calculating the probabilities will be roughly equal. This gives only an average of two reneging examples per group, but the pattern of reneging actually gives quite a different picture.

Position Class	Av. Service Time Range	N° of Examples	N° of Reneges
0	0 to 0.749	45	0
0	0.75 to 1.249	54	2
0	1.25 to 1.999	51	0
0	2.0 to 2.749	47	0
0	2.75 to 4.999	52	0
0	5 and above	40	2
1	All	59	11

Table 6.15 : Group Ranges for Reneging Decision Data

The data set that was formed can be seen in Appendix I, while the groups are summarised in Table 6.15. As can be seen, the majority of the data examples are in queue position class 0. To maintain large enough groups, the entire range of service time averages was used for queue position class 1.

The actual probabilities were determined by the position of the reneging examples in the sorted group, and the number of items in the group. A running total for the probabilities was kept, being incremented (by 1/Number of examples in the group) each time a new reneging case was reached. Table 6.16 shows an extract from the data set demonstrating the probabilities.

Position Class	Av. Service Time	Prob. of Reneging (Cumulative)	Prob. of Not Reneging
0	0.99	0	1
0	1	0	1
0	1.01	0	1
0	1.01	0.0185*	0.9815
0	1.1	0.0185	0.9815
0	1.1	0.0185	0.9815
0	1.14	0.037*	0.963
0	1.2	0.037	0.963
0	1.2	0.037	0.963

* Reneging Example

**Table 6.16 : Extract from Reneging Data Set Showing Cumulative Probabilities
(Group shown has 54 examples)**

6.5.7.4 Developing the Neural Network Model

The choice of neural network approach to use was determined primarily by the nature of the data. The OUTNET approach can form ordinary, but not cumulative probabilities and so cannot be used. The INVAR approach could be used, but because the data set uses all of the values of the key variable, it is likely to be bigger than one which uses a set of classification groups (as used in the previous models). This means that a data set with a random number variable added would require a set of probability values to be added for each data point and so is likely to be very large, with the result that training will be very slow. In addition, the small probabilities involved in the data make the accuracy of the approach low unless a large number of probability values are used. The OUTDATA approach allows the cumulative probabilities to be represented as target outputs, and does not increase the size of the data set. Therefore OUTDATA was the chosen approach.

The choice exists to represent each decision group as a separate neural network or to use a single network. It was hoped that a single network would allow some smoothing and interpolation between the decision groups. It also has the convenience of representing all the relationships inside one model.

The data was scaled between 0 and 1 for all of the variables. In fact the only variable that needed re-scaling was the average service time which was scaled from the range 0

to 15 (covering all the data ranges plus about another 30%). Networks were trained with 4, 6 and 8 hidden nodes.

The networks were tested using 1000 randomly generated cases. The results are shown in Appendix I. The networks with 6 and 8 hidden nodes produced very similar results, with those for the 6 hidden node network being shown in Figure 6.18. The results for the 4 hidden node network were similar to the others for the queue position class 0, but in the class 1 case, the extrapolated line turned downwards when the average service time reached around 6.5, rather than being cumulative.

The 2*6*2 network was chosen since the results produced were little different from the larger network, and having fewer nodes and weights means that its use in terms of memory and computation speed is better.

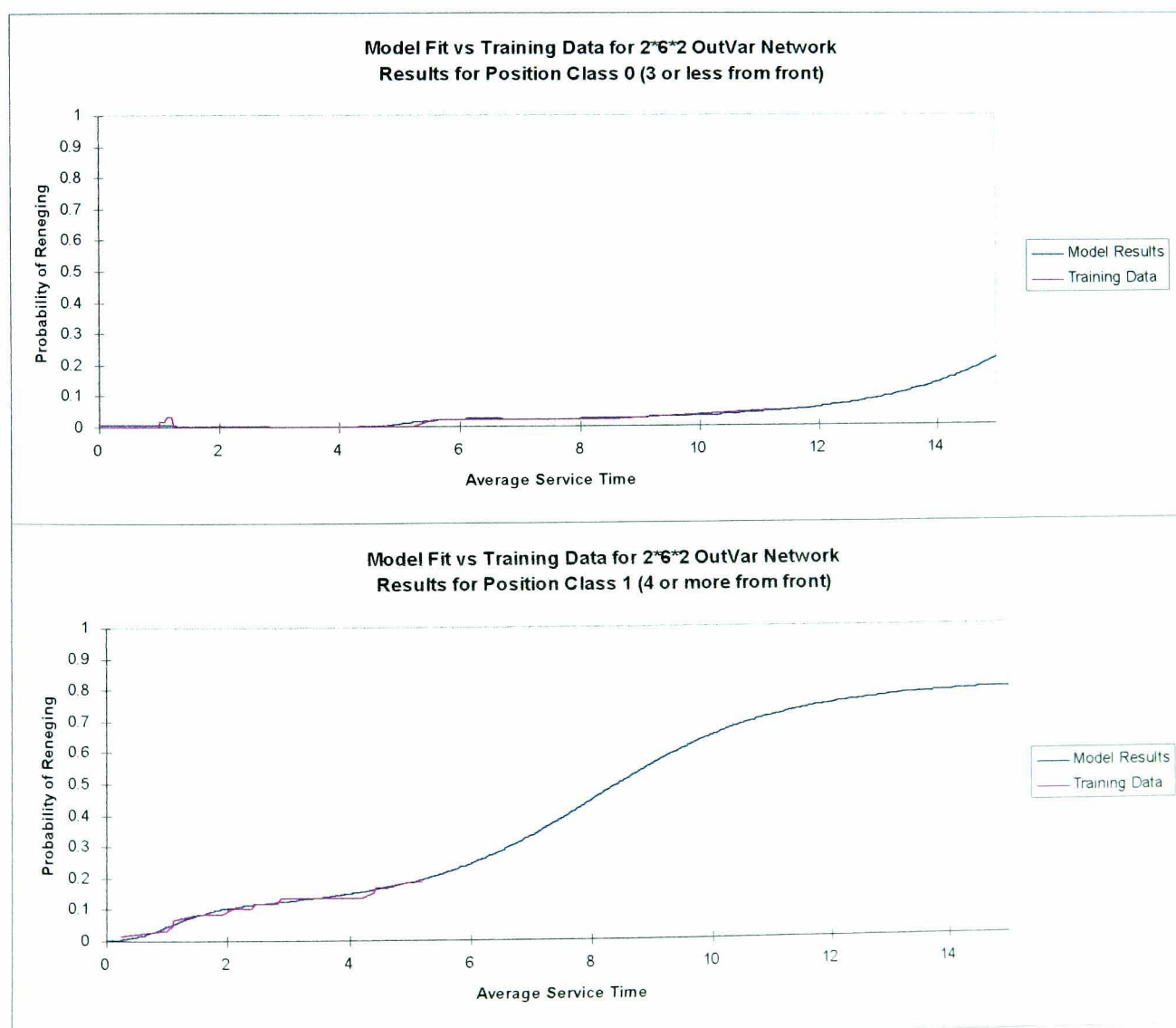


Figure 6.18 : Renege Model Test Results for 2*6*2 Network

The graphs in Figure 6.18 show the structure of the model which has been visualised using the test data, and compare these against the training data for each of the two queue position classes. These are plotted as lines rather than showing the individual data points for ease of comparison.

For the position class 0 case, the probabilities in the training data are all small. At the lower end of the average service time, the test probabilities are close to 0, with the network almost completely ignoring the small blip in the training data. In fact the probabilities move in a downward rather than an upward direction which is counter to the notion of the cumulative probabilities. After that, the network fairly closely (but smoothly) follows the slow rise in the probabilities, continuing to extrapolate past the training data with an increasing probability of reneging. Not shown on the graph (to maintain the detail of the main body of the distribution) are the results for very large queues. In fact that probability of reneging increases steadily, leveling out towards average service times of 30 minutes, with a reneging probability of 81%.

The model for the queue position class 1 case follows the training data fairly well, with a smooth fit. Extrapolation past the training data results in increasing probabilities of reneging, tailing off towards an average service time of 15 minutes and eventually levels out with a reneging probability of 81%.

The lower end of the queue position class 0 model is not reliable, with decreasing probabilities, although these are small. In this part of the decision model, with customers close to the front of the queue and small average service times, very little reneging would be expected. There can be considered to be no reneging at this point of the decision model without little loss of accuracy, although it does ignore a small blip caused by two cases of delayed balking. The shape of the rest of the model for both classes is smooth, and follows a shape that matches the intuitive expectation of increasing rates of reneging as the average service times increases.

6.5.7.5 Rule-Based Components

The use of seven decision groups requires some rule based-component for the sequencing of these groups as the queuing conditions that customers face move them between groups. There is also the issue of overriding the poor model shape for queue position class 0 cases where the average service time is low (0 to 3.99). This effectively removes the use of the neural network model for these decision groups.

In dealing with the decision groups, a look at Figure 6.16 indicates that only 3 decision groups need to be considered. For queue position class 0, the customers with average service time between 0 and 3.99 will be considered to have no chance of reneging. For average service times 4.0 and upwards, the neural network model has formed a smooth cumulative function which can be regarded as a single decision class requiring the sampling of only one random number for each customer. For the queue position class 1 case, there is only one decision group, and again this requires only a single random number. Table 6.17 summarises the decision groups and the approach used to resolve the decision.

Decision Group	Queue Position Class	Av. Service Time	Decision Model
A	0	0 to 3.99	No Reneging
B	0	4.0 and above	Neural Network
C	1	All	Neural Network

Table 6.17 : Summary of Decision Groups and Decision Models

Changing between decision groups is an issue. The position of a customer within a queue can not get worse, so a customer who is in Decision Group A or B cannot change to Group C. However, customers in Group C can change to A and B. This creates the possibility that a customer in Group C who has stayed in quite a long queue, with potentially high average service times, may end up reneging when they move into Group B because of sampling a new random number. This does not make sense as part of the decision process, but is a side effect of the framework.

To deal with anomalies caused by changing between decision groups, a rule was added such that the highest average service time will be recorded for a customer, and when moving between groups, a new random number will only be generated if that highest average service time is reached again. This prevents problems of a customer reneging when queuing conditions have actually improved, and also prevents the repeated re-sampling of random numbers if a customer is oscillating between two decision groups (only possible for Groups A and B in this case).

An additional rule in the decision is that a customer who is the next to be served will not renege. In the data set there are no instances of reneging when a person is in this position in the queue. Although more data would be required to have a more definitive view on this case, experience and intuition suggest that reneging under these conditions would be extremely unlikely.

In summary, the rules to be applied are :

- Only consider reneging decision where average service time has exceeded previous maximum for that customer
- If next to be served then no renege
- If within 3 of front of queue and average service time < 4.0 minutes then no renege
- If within 3 of front of queue and average service time ≥ 4.0 minutes then consult neural network model
- If further than 3 from front of queue then consult neural network model

6.5.8 Queue Swapping Decisions

Customers have the opportunity of swapping queues where more than one counter for a service is open. The exception is the Information service which employs a single queue system.

The data collection approach represents a poor level of realism for this decision since the human controlled customer is the only one that is allowed to swap queues. Therefore they face no competition for joining a shorter queue, except with new

customers entering the bank. Also there are no spatial issues of having to pass by several other queues to swap from one to the other. In practice a customer's decision would be affected by the risk of leaving their queue, crossing over to the shorter queue, and then someone beating them to it, resulting in them being in no better, or even a worse position.

6.5.8.1 Queue Swapping Data

The data containing information about queue swapping is the same as that which contained information about reneging. Table 6.18 summarises the number of cases of queue swapping that were observed.

Service	N° Customers Swapping Queues	N° of Customers Joining Service	% of Customers Swapping Queues
Information	0*	100	0
Transactions	34	139	24.5
Business	11	55	20
Currency	5	53	9.4

* Information is a multi-server single queue system

Table 6.18 : Summary of the Number of Queue Swapping Cases

Table 6.18 shows that Transactions had the highest incidence of queue swapping, with nearly a quarter of all customers swapping queues. Business was close behind, with a smaller percentage of queue swapping for the Currency service. The Transactions service has the largest number of counters that could potentially be open, so it is not surprising that it has the highest rate of queue swapping. The difference between Business and Currency is more surprising since they are organised in a similar way, although it could be down to a higher rate of reneging for Currency (effectively reducing the pool of customers), and also just differences in the scenarios.

The data was analysed to find the queue conditions at the points where customers did swap queues, and also conditions where customers could have benefited from swapping queues. The results of this analysis are shown in Table 6.19. It clearly shows that in the vast majority of cases customers swapped queues when the opportunity arose, a few swapped when they made no progress in queue position (probably because they were in a queue that was not making fast progress), while there

were only a few cases where the opportunity to improve their position was not taken. The currency counter has the largest proportional incidence of not taking up opportunities to swap queues, although with the small number of opportunities observed, there is little data for developing a model.

Service	Changed Queue		Did Not Change Shorter
	Shorter	No Shorter	
Transactions	32	2	1
Business	9	2	0
Currency	3	2	3
Total	44	6	4

Table 6.19 : Analysis of Queue Swapping and Queue Swapping Opportunities

With the weakness of the data collection framework for producing a general model involving queue swapping where there is competition between customers, and with fairly high consistency in the decision making, there is little to be gained from building any sort of neural network model for customers. From the results, it seems that the main question is not whether anybody will change queues, but who ? To develop such a model using a data collection framework similar to the one used here is likely to require an iterative approach. Some competitive behaviour could be added to the model to observe the human controllers' decision making, and then the competition model refined in light of the results, that model used in a further stage of the data collection framework, and so forth. To truly get a more representative model would require either a multi-user framework, or direct observation of queue swapping behaviour.

6.5.8.2 Rule-Based Decision Model

For the bank simulation, due to the lack of data on competitive behaviour, a rule-based model was be used to model the queue swapping behaviour of customers. These rules mainly arbitrary, and fairly simplistic. In developing the rules, the following considerations were held in mind. Customers are only likely to swap to a queue that is next to them, although the counters are close enough together that counters which are closed can be ignored when determining neighbouring queues. If customers have a

sense of justice, then from all the customers who would benefit from changing queues, it would be the person who has been waiting longest who would be allowed to swap. It is observed however that such altruistic behaviour does not always take place.

From the data, it is noted that when the opportunity to change queues arose, 44 out of 48 people decided to change queues. Therefore, the probability that any individual will change queues, provided they would gain from it, is estimated as 91.67% (44/48). The possibility of swapping where there is no gain in terms of queue position is ignored.

Selecting one of the neighbouring queues at random (only an issue with Transactions which might have more than two counters open), the first person in the queue who would benefit from swapping is given the opportunity to swap (a random number sampled and compared against 91.67%). If that person does not swap, then the equivalent position in the other neighbouring queue is tested, and so-on, moving back through the queues until all of the customers who might gain have been tested. Note that if a customer does change queues, the customers further back in the queue might still gain by swapping (for instance if a new counter is opened up), so this is considered.

If changing queues offers the opportunity to be served immediately (the other counter has no queue), then if no-one in front has changed queues, the person at the back of one of the neighbouring queues (the first one of these tested) will always swap queues, since they have little to lose by changing. Situations may occur, where there is a free counter, but no-one in the neighbouring queues. In this situation, the customers at the back of one of the next nearest queues will swap.

The model does offer some random variation in behaviour, although the probability of any one individual swapping is high, so the behaviour will be fairly consistent. It takes account of the relative positions of the queues, and it assumes that those people who have been queuing longest will at least have an opportunity to change. The rules also take account of cases where a new counter is opened.

In summary, the rules are:

- Only those customers who would benefit in terms of improving their position in the queue by changing will be tested
- Only those people in neighbouring queues will be tested
- A customer who is tested will have a 91.67% chance of swapping queues
- If more than one neighbouring queue exists, one shall be selected at random to start the testing
- If the tested customer does not swap, the next eligible customer will be tested. If another neighbouring queue exists, these will be tested alternately, otherwise the next eligible person is the next one back in the queue
- If the opportunity exists to get served immediately by swapping queues, then if no previous customer has changed, the customer at the end of the neighbouring queue shall do so. If two neighbouring queues exists, the first person tested who is at the end of a queue shall change. If there is no-one in the neighbouring queues, then a customer at the end of one of the next nearest queues will change.

6.6 Implementing the Decision Modules

The neural network and logic models were created in the previous section. These were then coded up and implemented into the bank simulation so that decision making is made by customers without human intervention. The aim of the implementation was to test that the full decision modules work in the simulation environment, and to identify issues of interfacing the decision modules with the rest of the simulation code. Full implementation details and code can be seen in Appendix J. Here an overview of the implementation strategy, and the decision modules will be given.

6.6.1 Implementation Strategy

The strategy used was to implement the decision modules in two parts. Firstly, the modules were implemented for only the special customers (those for whom the decisions were made by the human subjects) and were used within the data collection framework. The second part of the implementation involved applying the decision

modules to all of the customers in the simulation, and to remove the data collection framework, scenarios and special customers, leaving only the basic simulation.

All the coding was done using Borland Pascal for DOS. This was to match the simulation software. A Prolog implementation of the logic parts of the decision module would have been an interesting exercise, but the difficulty of interfacing two DOS based packages inhibited this. As it was, the logic was sufficiently simple that coding in Pascal was relatively straight forward.

6.6.1.1 Implementation for Special Customers

The decision modules were implemented bit by bit, with testing carried out at each stage. Concentrating on only the special customers made the process easier since these could be easily observed. Data was collected and displayed about them to ensure that the decision modules were operating correctly.

In the initial stages of implementation, tests rigs were created to check that the neural networks were running properly. This enabled input values to be entered into the networks and the response observed. Once it was assured that the networks were operating as expected, the decision modules were implemented one at a time, at each stage maintaining the full functioning of the simulation. The behaviour of the special customers could be observed, and the inputs and outputs of the neural networks were displayed. This ensured that the decision modules were operating with the correct logic, and that the neural networks were receiving the correct information at the right time, and producing the expected results.

Figure 6.19 shows a screen shot for the implementation of decision modules for the special customers. In this shot, the special customer is queuing for the Information service.



Balking Response

Reneging Response

Figure 6.19 : Screen Shot of Model with Decision Modules for Special Customers

At the bottom of the screen on the left is the neural network data for the arrival decision at the point that the special customer arrived at the bank. Reading downwards, for the point at which the special customer arrived this shows: the number of Information counters open, the number of customers in the queue, the random number, and then the neural network responses for joining the queue and balking. On a winner takes all basis, the decision is to join the queue. Note that the state of the queue shown in the screen shot has changed since the special customer entered the bank.

On the bottom right-hand part of the screen, the data for the reneging neural network is shown. This displays the information for the point just before the queue moved. Reading downwards this shows: the position of the customer in the queue, the average service time, the random number of the customer, and the neural network responses are the probability of reneging and of not reneging. The random number for the

customer is compared against the probability of reneging, if it is smaller then the customer leaves. In this case the customer stays, but later on in the scenario the probability of reneging increases above 0.21 so the customer leaves.

6.6.1.2 Implementation for All Customers

After the implementation for special customers, the simulation was adapted so that *all* the customers used the decision modules for their decision making. This involved removing the data collection framework, including the use of scenarios and special customers. The actual decision modules required virtually no changes for use with all of the customers, only a few inhibiting clauses were removed which had previously been used to ensure that decisions were only made for special customers. Some additional coding was required for the simulation to capture information for each of the customers that was required for the decision making, and to actually call the decision routines from the simulation code. The neural network data display was kept until it was assured that the decision making modules were functioning correctly for all the customers.

The resulting model was the basic simulation with the addition of the decision modules. A day framework was added to the model, so that the simulated bank was open for specified hours in the day, with thinning used for the arrival rates. An option was then added to allow the user of the simulation to interact with the model by changing the positioning of the staff in the bank. This allowed staff to be placed at the beginning of the simulation run, and then while the simulation was running the staff members could be allocated to other counters. The staff member is actually moved when they have served everyone in the queue (no more customers have been allowed to join) and have cleared up the current counter. The clearing up process follows a Normal distribution with mean of 5 minutes and standard deviation of 0.5 minutes.



Figure 6.20 : Screen Shot of Model with Decision Modules for All Customers

Figure 6.20 shows a screen shot of the full implementation of the decision modules for all of the customers. The yellow colour for the counters indicate that the Transactions counter is about to be closed down so that the member of staff there can open an Information counter. The table on the left-hand side of the screen shows the percentage of customers for each service who have bailed or reneged.

6.6.2 The Neural Networks

The first stage of implementation was to load all of the neural networks into memory at the start of the simulation run. The network code itself is a very much cut down version of that used in the neural network training software. The networks are constructed of linked lists which allow the network structure to be created. The node biases and weight values that were developed in training are loaded into these structures. These networks are able to accept input values, and produce the appropriate responses, but cannot undergo further training. The technicalities of the neural networks are hidden in the Simnet program unit. In the simulation program

there is a *nnet* structure type, and a variable of that type is declared for each neural network to be used. There are four commands related to the neural networks: to load the network into memory, to enter an input value (needs to be called for each input), to operate the network (i.e. to produce a response), and to read the network output values (called for each output). One of the parameters for each command is the variable name of the neural network, allowing several networks to be used concurrently in the same program.

In total there were five neural networks, a balk/stay network for each of the service types, and a renege network. A test rig was written that allowed values to be entered into the networks, and the network response displayed on the screen. The results were compared with the responses from the test data that was used in developing and evaluating the networks, as described in Section 6.5. All of the networks produced the correct responses, and no problems were encountered regarding memory or interference between the networks.

6.6.3 Arrival Decision Modules

The arrival decision modules were developed one at a time for each service type. However, the principles in setting up the module for one service largely held for the others. The main differences (other than the different neural network structures) were the rules for deciding which queue to join.

The arrival decisions for each counter type were implemented using three sub-routines (except for Information which has only two). One deals largely with generating a neural network response. This receives parameters for the number of open counters for the service, and the size of the shortest queue. The parameters are scaled using the same ranges as those used for the original training data, and the scaled values are applied to the network as inputs. A random number is generated for the third network input. The network generates two outputs, a staying value and a balking value. The output values are compared, and the greater of the two determines the decision. The decision result is returned as the output from the routine.

Another routine determines which queue the customer should join (this is not done for Information since it has a single queue). The locations and number of the shortest queues are checked, and the actual queue to be joined is determined using the rules developed in Section 6.5.6.

A master sub-routine controls the sequencing of the decision making. The first step is to check that a neural network based decision is required (customers automatically stay if a server is free), and if so, the neural network routine is called to make a decision. If the customer does stay, the routine called to determine which queue is joined.

Finally, a general arrival decision routine was created which looks at the details of the customer and calls the decision module for the appropriate service.

The neural network sub-routines are stand-alone, and communicate only with their respective rule-based controllers through the exchange of parameters. The rule-based routines interact more directly with the main simulation by collecting information about the system state. The simulation state is only affected by the decision modules through the output of the decision by the general arrival decision routine, and even then this is only in terms of what use the simulation makes of the decision outcome.

6.6.4 Reneging Decision Module

A single reneging module serves all of the service types. Despite some of the difficulties in developing the initial model, as described in Section 6.5.7, the implementation of this module was actually the most straight forward of the three types. Some extra effort was also needed to make the entities store the required information for the reneging decision. This required minor changes to both the simulation model, and the simulation development software (to store and deal with real valued attributes for entities). The customer entities are required to store attributes for their position in the queue when they first joined the queue for the service, the simulation time when they first joined the service, and the highest average service time that the customer has experienced. A fourth attribute stores the queue position class

when that customer last checked the renege module. This is used to determine if a new random number needs to be generated for the customer.

The renege decision module is implemented in two main routines. One is the neural network routine which takes the decision parameters as inputs (queue position class, average service time, and random number). The neural network inputs of queue position class and average service time are scaled and applied to the network. The responses are the probability of renegeing and the probability of staying. These outputs are rescaled to ensure that they sum to one (although experience with the test data suggests that little re-scaling is needed). The renegeing probability is compared to the random variable to see if the customer reneges or not.

The other routine contains the control logic for the decision. The state of the system and average service time for the customer are checked against the conditions for renegeing to occur, as described in Section 6.5.7. The neural network is only used if the appropriate conditions hold. A new random number for the customer is generated in only two cases: if the customer is checking the renegeing network for the first time, or if the customer queue position has changed from class 1 (4th or further back) to class 0 (3rd or closer to front), otherwise the old random number is retained.

The neural network routine only communicates with the rest of the model through the exchange of parameters. The control routine interacts with the rest of the simulation by examining the system state, and directly changes the attributes of customers for the random number store and the position class.

A third routine acts as the main interface between the decision module and the simulation by calling the decision module for a queue of customers, and determines their actions in response to the decision results. One issue raised in Section 6.5.7 was to determine the points in time at which the renegeing decision should be checked. In the implementation, the renegeing decision is checked for all the people in a queue when it is about to move forward (i.e. the service for the person at the front is about to finish). This represents the point where the average service time is at its worst since

the queue last moved. This matches the situation of the data that was used to train the network. The queue is checked from last to first, so that the decisions of customers are not affected by other customers reneging in the queue in front.

Using the change in the state of the queue as a basis for checking reneging is always going to be an approximation of a continuous process. There are possible side effects of using the approach in terms of increased balking. A customer who arrives between changes of queue state might be induced to balk by the queue size, when they would have stayed because of a shorter queue due to a customer in reneging just before they arrived (the reneging will not be determined until the queue is about to change state and so as far as the simulation is concerned it will not have happened yet). However, the incidence of reneging is likely to be extremely low so that the minor effects of this anomaly are far outweighed by the computational advantages of the approach.

6.6.5 Queue Swapping Modules

Queue swapping modules were created for all the service type except Information. These contained no neural network component, being rule based with a stochastic component. These modules were the most conventional in terms of ordinary simulation code, although implementing the logic was quite complicated. The queue swapping decision was checked every time a queue became shorter and involved looking at the neighbouring queues to see if customers could benefit from changing.

In many senses the decision making component was simply sampling a random number to determine if a customer would swap queues. The complexity lay in determining who would be making the decision.

The coding for the Business and Currency counters was relatively straight forward since there were only two counters to be concerned with. These were implemented using only one sub-routine each. The Transactions service was more difficult because up to five counters could be involved, and was implemented using six sub-routines. Implementation involved finding which queues were the nearest open neighbours to the queue of interest; dealing with cases where a queue has one or two eligible neighbours;

the shuffling effects caused by somebody from a queue swapping, and so providing customers in other queues the opportunity to gain a position by swapping to the queue that has been vacated; and customers swapping from non-neighbouring queues when there is a free counter and no-one in a closer queue to join it (the other customers already being served at the time).

The decision modules were closely coupled with the simulation code, collecting data about the simulation state, and also controlling the actions of customers. Such interactions in the decision module are not ideal, but due to the highly procedural nature of the code, were largely unavoidable.

6.7 Observations on the Bank Simulation

The following are some observations and reflections on the use of, and experience gained from the Bank Simulation. This looks at the strengths and weaknesses of using the Bank Simulation as a vehicle for investigating the stochastic neural networks, and notes some of the specific issues raised by the exercise. Chapter 7 contains a full discussion of the stochastic neural network approach and the hybrid model.

6.7.1 The Bank Simulation as a Vehicle for Investigation

Section 6.1 discussed the difficulty of having access to, and collecting decision making data, and made a case for the use of the Bank simulation as a suitable vehicle for testing out the stochastic neural network approach in practice. Using a computer-based medium for data collection is always going to lend an air of artificiality to the exercise, although this will be governed to some extent by the complexity of the hardware and software used to create the environment. In this case the Bank simulation was a relatively simple environment, particularly in comparison to some of the military flight simulators. However the Bank simulation was accessible to a general user, and represented something that they could relate to. The graphics, while simple, were clear and understandable, and offered no problems to the participants who provided the decision making data. Within the confines of the environment, and data collection framework, the Bank simulation does provide a valid source of decision

making data for a first stage of examining the use of the stochastic neural network approach.

The Bank simulation has its disadvantages. All of the decisions involve discrete outcomes, either yes/no decisions, or queue locations. In the end the neural networks were only used for the balk and reneging decisions which were both yes/no decisions. Also creating a model with a number of service types, each with several decisions associated with them, increased the number of aspects of the model that needed data to be collected for them. With a limit to time available and the number of volunteers to use the simulation, this inevitably limited the amount of data that was collected for each decision.

An alternative approach would have been to use several simulation models, each with a very restricted focus on the decision that needed to be made. This would have allowed a greater variety of decision situations to be examined, with different aspects to the variables. Having more focus on the decisions to be made would have allowed more data to be collected, so that more dimensions could have been modelled for the decision criteria and/or some of the collected data could have been used for testing purposes.

On the other hand, the Bank simulation had some advantages. Trying to create decision making models for a whole situation forced the consideration of areas of decision making that might otherwise have been avoided. The consequence of using very focused decision situations would have been a choice (consciously or unconsciously) of situations in which the use of the stochastic neural network approach would have been more straight forward. The bank simulation represents a more likely situation in practice, with a number of decisions to be made, some difficult issues to deal with and limited amounts of data.

6.7.2 Decision Models

The Bank simulation offered the opportunity to attempt to model three main decisions: the decision upon first arriving at the bank, the renege decision and the queue swapping decision.

The arrival decision was a demonstration of how a highly specific problem could be made more general, and so making better use of the data available. This resulted in the splitting of the decision into two: the balking decision and the queue joining decision. The balking decision was successfully modelled using the neural network approach, despite the problems of having fairly limited data. In particular, for many of the service type there were issues of incomplete data for the tails of the distributions which had to be tackled. In the end, all of the chosen models used the INVAR network approach.

The queue joining decision was more difficult to deal with since the outcome was queue specific. The number of dimensions in the Transactions case made the problem of dealing with many different combinations of possible counter states insoluble without very large amounts of data. For the Business and Currency counters, with a maximum of two counters open, it was the opposite case, that the problem was too trivial for the use of the neural network approach.

The renege decision provided a good opportunity to examine the modelling of a situation with a continuous time dimension in a discrete event simulation. The issue was complicated further since the simulation did not try to approximate a continuous system by using fixing time increments, instead the time was only advanced when the system state changed. The modelling solution required a re-think of the approach used in Chapter 4 with the artificial data. The approaches there could be applied to one off decisions (as far as an individual customer was concerned) at clearly identifiable points in the entity cycle. The renege decision did not have clearly identifiable points in time, and involved repeated applications of the decision making model for an individual customer. Repeatedly sampling a random number for the decision would have resulted in a lack of consistent behaviour for an individual customer, and would have made the

probability of reneging connected to the regularity with which the decision model was checked.

The solution for the reneging decision involved training the neural network to represent a continuous probability function, this overcame the issue of consistent behaviour, and provided that the decision is checked when the queue state changes, the issue of the regularity of checking the decision model. It did however raise issues in representing the continuous distribution across several dimensions and where the data examples are not evenly spaced. The use of decision regions to split the problem space into more homogenous and easily manageable groups was not an ideal solution, but it was a pragmatic one that at least reduced the problems of representing the continuous distribution. The choice of neural network approach for representing the situation was limited by the nature of the data model, but one of the approaches investigated in Chapter 4, the OUTDATA model, was applicable.

The queue swapping decision was the most disappointing in terms of testing out the stochastic neural network approach. The data collection framework involving one subject at a time was severely limited in its ability to create useful data for representing a situation that has a high degree of interaction and competition between customers. While queuing provides a structure and limit to the behaviour of customers in relation to each other, the queue swapping breaks that structure. The nature of the problem and the type and amount of data collected made the use of a neural network untenable. Thus it did little to investigate the modelling ability of the network approach, although it did provide some information on the types of decisions that are difficult to deal with, particularly using a simulation framework for data collection. It also provided some interesting comparisons in the ease of implementing a neural network based decision module in comparison to a more conventional procedural one.

6.7.3 Decision Model Validation

Pidd (1992) notes that model validation is seen an important part of the simulation development process, but is also one of the most difficult ones. Validation is an

important part of creating a credible model, and this holds as much for the decision making models as for any other part of the simulation.

The validation of decision making models in the knowledge-based systems area is an equally difficult one, and validation involves comparing the performance of the system against that of a human. In general, there is the ability to determine (in retrospect) whether the decision made was correct or not, and so performance can be measured. Decision areas where performance is poor can be investigated and the rules refined. Finding the rules which cause problems can be difficult, and due to the large combination of possible rules states, complete validation of the model is unlikely to occur. In the case of the stochastic neural networks, the decisions are not designed necessarily to be optimal, but to represent the variety of decisions by humans. This means that a single decision cannot be compared against that of a human, but in fact a whole set of decisions with similar decision criteria must be compared to see if the spread of responses is correct.

The stochastic neural network is more akin to an empirical input probability distribution, albeit a rather complicated type. The network has the added responsibility of smoothing the model and interpolating between the data points. The problem of validating the model is increased by the effects of choosing different numbers of nodes for the hidden layer of the network, as shown in Chapter 4 and the experiments described here. As is the case for ordinary empirical probability distributions, there is no theoretical model to compare against, and so the only form of validation is to compare the model against more data from the generating process. Removing some of the data for validation clearly has an impact on the modelling since less data is available for model building. This is particularly acute for the decision models since the data is required for several dimensions and interaction of data values across the dimensions is important. This means that the modelling and validation processes are both extremely data hungry.

The purpose of the above discussion is to demonstrate that validation is a difficult issue in general, and particularly so for decision making models, both rule-based and those

using the stochastic neural network approach. The difficulty of validation is a weakness of the approach, but it is not unique to the stochastic neural network approach.

Some sort of validation is certainly required, no matter how limited the data is. A choice between alternative models is required, and a view on the reasonableness of the final model to be implemented is necessary. The approach taken in the experiments here was to at least visualise the decision models by generating artificial decision criteria and observing the neural network responses. This allowed a view as to whether the results produced were reasonable, credible and internally consistent. In the absence of data to do a more rigorous analysis of the models, it is argued that the approach taken was a reasonable one, if subjective, and reflects the issues faced in a great many cases for potential practical applications of the technique.

In terms of evaluating the stochastic neural network approach, the situations modelled in Chapter 4 where theoretical models were available for the testing the results, and the situation in the Bank Simulation where no such theoretical models were available are both important aspects to be considered.

6.7.4 Decision Model Implementation

In many respects the issues of decision model implementation into the simulation environment are the least important for this investigation. It is encouraging that the neural network based decision modules could be implemented with relative ease. The environment used is a programming environment, which helps with the task.

The implementation of the approach in other simulation environments, object oriented approaches and model building in multi-application environments are all issues in the practical application of the approach for specific situations, but not in the discussion of the principles of the approach. It is the principles that are of concern in this study. If the principles are thought to be a good and worthwhile, then the technical aspects of the implementation for a particular case can be addressed as required.

Chapter 7

Discussion of the Stochastic Neural Network / Hybrid Modelling Approach

This chapter draws on the insights gained from the study, discusses the main outcomes and lessons, and considers the general application of the stochastic neural network approach. Section 1 reviews the aims of the research in order to set the discussion into context. Section 2 considers the theoretical basis of the stochastic neural network approach. Section 3 discusses the main outcomes for the experiments with artificial data sets, while Section 4 discusses the lessons from the application of the hybrid models in the Bank simulation. Section 5 outlines the main strengths and weaknesses of the approach, and compares these with the rule-based and abstraction approaches. Section 6 considers the features of application situations that would make the use of the stochastic neural network approach worthwhile, and suggests some application areas.

7.1 Research Context

Before starting the discussion it is useful to briefly review the main aims and context of the study to understand the key issues to be addressed.

The original brief of the study was extremely broad, involving the general investigation of approaches for representing intelligent decision making in discrete event simulation. Of particular interest was the concept that people use different decision making processes with the result that decisions can vary, even when exactly the same circumstances apply. Coupled with this is the notion that people do not always (in fact rarely) make optimal decisions and that there are circumstances in which we want to represent this sub-optimal and variable decision making. Neural networks were recognised as an approach that, in theory at least, might provide a way forward for investigating this type of decision making, and so provided a focus for the research described in this thesis.

The representation of intelligence in simulation models is a valid area of concern for simulation analysts, but it does tend to fall outside the mainstream of research. Consequently there is some but not an abundance of past research to draw on. The representation of variability, and sub-optimal behaviour is an area that has had relatively little research done. The use of neural networks in this area, particularly with the consideration of stochastic elements represents a new direction of approach. The work by Hurron (1993) in using neural networks to represent input probability distributions provided a marker for starting the investigation, but the representation of intelligent decision making provides a whole new set of circumstances with which to deal.

The research was exploratory, with the aim to assess the feasibility of the stochastic neural network approach. A step by step approach was taken to make sure that the decisions made at each stage were as informed as possible. The study aims to answer some basic questions about the approach, but also to raise further questions that are of a more informed nature than at the beginning.

7.2 Basis of the Approach

The key aspect of the stochastic neural network approach taken to represent intelligent decisions making is the ability to deal with variability, at the same time as taking account of criteria that might affect the decisions. Rule-based systems were seen as being weak in the former respect, while abstraction was weak in the latter. From the initial review of literature, the neural network approach had features that suggested that the goals could be achieved together.

The multilayer perceptron was chosen as a tool that might allow the modelling goals to be achieved. It was chosen over other tools because of its flexibility in handling different types of data (real and classification), and its features as a non-linear, non-parametric regression tool. It is one of the best known, and most researched types of neural networks and so provided a solid vehicle to use in undertaking the research. Conceptually the view of the multilayer perceptron taken here is more as a statistical

tool than an artificial intelligence one, despite its application for representing intelligent decision making.

In terms of key principles, it is the formulation of the stochastic representations that is more important than the specific regression tool for fitting. The flexibility of the multilayer perceptron makes it a good contender for applications, but it is possible that other regression tools offer advantages in terms of speed of training or accuracy for data with specific features. For instance, Cheng & Titterton (1994) review some developments in the area of statistical modelling in a comparison with neural networks, while Wang (1993) proposes an alternative neural network for fitting monotonic non-linear models.

Three possible formulations for representing decision making with stochastic elements were suggested. The INVAR model uses a random number variable as one of the inputs, much as in ordinary empirical distribution fitting, but expanded up to include several input dimensions. This model can be used for cases that have either real valued or classification values as outputs. The other two models can only be used where the decision outcome is in the form of a classification. The OUTDATA model represents the probabilities for choosing each of the classifications as values in the target output data for various situations. The neural network is trained to associate particular values of the decision criteria with these probabilities. The OUTNET model is trained with target values of the actual decision classifications that were observed, with the neural network being allowed to determine the probabilities itself during training to account for variability in the observed decision making.

The above models were conceived for cases of one-off decision making. An adaptation of the approach is required when an entity is required to re-evaluate its decisions at various points in the simulation. Here there is a requirement for consistent behaviour for an individual, where the decision would only change due to changing circumstances. A probability distribution is still used for this situation, but this time it is a cumulative distribution with a random number being sampled and retained for an individual entity.

Data plays an important part in the modelling approaches, whichever specific model is used. The neural networks are data driven tools, and so the method is dependent on having suitable data. The more dimensions to the decision making criteria, the more data is required, particularly as the interactions between the decision variables need to be modelled. At the same time, the data might not be complete, having areas in the decision space where no data is available. The use of a hybrid framework, allowing the combination of both neural network and rule-based components introduces a flexibility to the modelling. Decisions can be split up into smaller sub-components requiring fewer decision criteria dimensions, and therefore less data, and rule-based components can be used for parts of the decision making where neural networks cannot be developed.

The discussion moves on to considering the approaches in the light of their use with artificial data, and in the application of providing intelligent decision making for customers in the Bank simulation.

7.3 Results from the Artificial Data Sets

The use of the artificial data sets allowed a comparison of the different stochastic neural network formulations, with the ability to test the results against known functions. The tests involved using the Beta distribution as a function with continuous outputs, and the Binomial distribution for discrete outputs. While Hurion (1993) used neural networks to learn how to represent input distributions with fixed parameter values, the experiments here varied the parameters values, so increasing the dimensionality of the learning problem.

For each data set there were further variations. Small and large data sets were used to investigate the impact of the amount of data available. Also there were versions of the data set that used the known probabilities from the distribution, and those that sampled randomly from the distribution and generated the probabilities empirically (as would be done in practice). This allowed an analysis of the loss of precision through the use of empirical probabilities.

Only the INVAR model could be used for fitting a model with continuous outputs. For the discrete data set, the INVAR, OUTDATA and OUTNET models were used.

Parameters	Data size	Probabilities	MAD	Data size	Probabilities	MAD
Beta($\alpha_1, 1$)	Small	Structured	0.007193	Small	Empirical	0.100149
	Large	Structured	0.002638	Large	Empirical	0.026842
Beta(α_1, α_2)	Small	Structured	0.005055	Small	Empirical	0.104835
	Large	Structured	0.005705	Large	Empirical	0.026446

Table 7.1 : Continuous Data Results Showing Mean Absolute Deviations (MAD)

The Beta distribution was a difficult case to model because the distribution becomes very sensitive for extreme values of the parameters. Firstly a data set which only varied one of the parameters was used, and then a data set was used that varied both of the parameters (with proportionally more data allowed for the extra dimension). Table 7.1 summarises the results of the experiments with the Beta distribution in terms of the Mean Absolute Deviation of the model results in comparison with the test data. In absolute terms, the accuracy of the representations was similar for both the one and two parameter cases. The structured data sets allowed considerably more accurate models to be developed, and the large data sets produced more accurate models than the smaller ones.

The Binomial data used had the number of trials fixed at 4, and so gave 5 possible classifications, while the probability of a successful outcome in each trial was allowed to vary. Table 7.2 shows a summary of results for all the modelling approaches using the percentage of correctly classified outputs from the testing data as the criteria for comparison. The OUTNET approach had two versions, one that divided the data set up into groups based on the value of the probability of success, and used the mid-point values of each group, and the another that used no grouping, and instead used the original values.

Small Structured		Small Empirical	
Approach	Testing % Correct	Approach	Testing % Correct
INVAR	84.55	INVAR	29.00
OUTDATA	98.85	OUTDATA	33.35
OUTNET	89.90	OUTNET(G)	21.85
		OUTNET(U)	69.00
Large Structured		Large Empirical	
Approach	Testing % Correct	Approach	Testing % Correct
INVAR	94.89	INVAR	87.25
OUTDATA	99.35	OUTDATA	80.45
OUTNET	91.25	OUTNET(G)	77.00
		OUTNET(U)	74.50

G = Grouped Data, U = Ungrouped Data

Table 7.2 : Discrete Data Results Showing Percentage of Correct Classifications

The results show that the OUTDATA approach is the best when the true probability values are known, producing extremely accurate classifications, but that it does not do so well when the empirical probabilities are generated. The INVAR approach was clearly more accurate for the empirical probabilities, seeming to be more robust to the inaccuracies introduced into the data set. All of the approaches did badly for the small data set with empirical probabilities except for the OUTNET approach with ungrouped values for the probability of success. It is thought that with very few data examples, the extra spread of values from using ungrouped data helped to fit a smoother and more generalised model through the data than for the other approaches.

It is a difficult task to interpolate across a multi-dimensional distribution, especially where the probabilities are empirically generated from the data. The results for the larger data sets show an acceptable degree of accuracy given the processes that were being modelled. The results for the smaller data sets with empirical data sets were not good. While the model had to do more interpolation between the training data points, this was also true for the small structured data sets, where the results were much better. With the data set being split up into groups, and with empirical probabilities being generated for each group, the small data sets allowed only very limited data for this. The results show that much more accurate models can be achieved when more data is available in each group to develop the empirical probabilities.

The results from the models for the discrete output data set suggest that where the probability estimates are accurate the OUTDATA network should be best. This is only likely in practice when very large amounts of data are available. Where there is likely to be more error in the probability estimates, the results suggest that the INVAR model would be better.

Overall, the methods are shown to be hungry in terms of the amount of data that they require to produce accurate models. However, in these experiments even the larger data sets were not particularly large, having an average of just over 10 data points per group with which to generate the empirical probabilities. In terms of accuracy, it would not be desirable to fall below this level of data. At the same time, using larger groups to have more examples in each group requires the model to do more interpolation between data points. Further experiments with fewer groups showed that the approach is probably less sensitive to having larger groups than having fewer examples in each group. The balance between the two will be dependent on the nature of the process being modelled. A process with a complicated decision surface will be more sensitive to the size of the group than one with a smoother decision surface.

In practice, the amount of data available will be a major factor in determining how many decision criteria can be included in the models.

7.4 Results from the Bank Simulation Study

The bank simulation study allowed an investigation of the practical implications of using the stochastic neural network approach by looking at a case study. The use of a simulation model as a framework for data collection made data accessible, but the number of decisions that needed to be modelled, the time and the number of subjects from whom to collect data from did limit the amount of data that was available. This restricted data availability is likely to reflect the situation for the application of the approach in many other projects and so has benefits in its study. It did however limit the number of decision criteria that could be modelled. In the end, all of the neural network models were trained using two decision criteria as the input parameters, plus the probability variable for the INVAR models.

The limited data and the spatial nature of the data involving queue choices required the splitting of the decision processes into parts, and the application of the hybrid neural network / rule-based approach. The actual interaction required between the neural network and rule-based components was fairly simplistic, and allowed the two to co-exist in the model without any problems.

The two main decision areas where the stochastic neural network approach was used were for deciding whether to stay or balk upon arrival, and deciding when to renege. Both have the same type of decision output which is a yes or no value, but the arrival decision occurs at a fixed point in time, while the renege decision contains time related aspects and needs to be re-evaluated for individual customers.

Issues faced in developing the neural networks that were not encountered with the artificial data involved having regions of the decision making criteria set with little data evidence and the problems of evaluating the validity of the models.

For the arrival decisions, there were a number of cases where there was little evidence of the queue sizes where the balk rate would be close to 100%. A number of the models coped well with this, and developed tails based on the general downward slope of the data in the main body of the distribution. In some cases artificial data was required to introduce points with 100% balking to force the distributions to tail off. This was generally effective, with models keeping the features of the main body and then tailing off to 0.

Evaluating the different models was a difficult situation because of the lack of data. With no known function to compare against, any testing data would need to be collected using the data collection framework, and would effectively reduce the pool of training data available. A substantial amount of data would be required for testing purposes. The models represent a distribution of behaviour, so that the testing data would need to contain evidence of the variability for evaluation of the distribution to occur. This amount of data was not available. Instead the validation approach

involved producing artificial input criteria so that the response of the network could be visualised and was judged on the basis of providing a smooth fit, internal consistency and plausibility. A number of networks were trained using different number of hidden nodes to model different levels of non-linearity, with the best being chosen on these subjective criteria.

Validation is likely to be a problem in all cases that data is not abundantly available. The use of visualisation of the model is an important step in the model building stage, even if its subjectivity means that it is not as rigorous as a full quantitative comparison with test data. In fact, visualisation should still be used along-side test data even if it is available.

Modelling the arrival decisions involved a process similar to that used for the artificial data. The earlier study had suggested the INVAR model as the best choice for use in practice, but the Bank simulation offered an opportunity for further comparisons between the approaches, and so all were used. In all the cases the INVAR models were selected for implementation. The models produced by the INVAR approach tended to be smoother than for the other approaches with a greater level of interpolation between the values of the number of counters variable, producing models that were internally consistent. On the other hand, all of the INVAR models had tails that reached zero probability, while the other approaches produced models with tails that had very low but not zero probabilities. However, it was felt that the advantages of the fit for the main body of the distribution outweighed the disadvantages of the closed tails. As suggested by the artificial data, the INVAR models seem less effected by inaccuracies caused by generating the probabilities empirically from sample data.

Modelling the renege process required a change from the approach taken for the artificial data. The decision needs to be reviewed at regular intervals. This means that the process will be called a number of times for an individual customer. Regularly calling the decision and sampling a new random number would have the effect of inconsistency in the decision making from the point of view of an individual customer. The random number can be viewed as a measure of the degree of tolerance that the

customer has when queuing. Regularly changing that tolerance level for a customer would be wrong. However, if the a customer does not decide to renege, they should be able to change the decision as circumstances change.

The approach to the renege model required sampling a random number and retaining its value for an individual customer to be re-used in the decision process. Since a multimodal probability distribution is possible, a cumulative distribution was used instead. The probability of renegeing at any point in time is conditional on not renegeing beforehand. As the conditions become more extreme, the probability increases and so brings into range people who did not renege before. The data needs to be grouped, except for one variable which is allowed to vary across its range and against which the cumulative probabilities are generated. The variable that is chosen to vary should have a high explanatory factor, and ideally should contain a time element.

Another issue faced during the modelling of the renege decision was the difference in the amount of data available for different levels of the decision criteria. This is a particular problem for the cumulative distribution since the effect of a few data points with extreme conditions and a high probability of renegeing is swamped by a lot of data points with low values and little chance of renegeing. This damps down the cumulative probabilities at the upper end of the distribution. To overcome this the continuous variable (average service time per customer while an individual is in the queue) was split into regions that were more homogeneous, and cumulative probabilities produced for each region.

The data that was produced was used to train a single neural network, with the hope that there were be some smoothing and interpolation effects between the groups. This was the case and in fact the model was smooth enough that some of the data regions could be amalgamated again for decision making, and so reduced the number of data groupings. The OUTDATA approach was used for modelling. The OUTNET approach is not capable of producing cumulative probabilities, while the INVAR approach would have resulted in a very large data set (through the inclusion of the

probability variable) and would have suffered with accuracy problems on the small renege probabilities that were a feature of much of the decision region.

Each group of data which had cumulative probabilities generated for it needed to be treated as a different decision group and have a different random number generated for it for an individual customer. A customer can only be in one group at a time, so only one random number needs to be stored at any particular time. Rules needed to be added to allow for the smooth transfer from one decision group to another, which involved only checking the renege decision if the average service time was greater than the longest time the customer had experienced while waiting in the queue. This meant that a customer would not renege when they moved to a more advantageous decision group (such as being near the front of the queue) purely because a new random number was generated at that point. In the end, there were only two decision groups that required the use of the neural network, and controlling the transfer between them was fairly straightforward.

The approach used to model the renege decision was a pragmatic one that dealt with the issues raised by the case in hand. The result was a plausible decision model for renegeing. However, there is no guarantee that the approach would be suitable for all cases of a decision with a temporal element. The features of the renege decision were the difficulties in the amount and spread of data available. In particular there were few instances of renegeing observed in comparison to non-renegeing. The nature of the decision outcome is that it is a yes/no response, with the customer leaving (and therefore making no further decisions) where there is a 'yes' decision. The number of decision criteria was low (queue position, and average service time of customers) with the average service being the variable factor against which to generate the cumulative probabilities. The resulting cumulative distribution was fairly smooth.

A number of decisions with similar factors to the renege decision could be envisaged. For instance, a driver wishing to pull out from a T-junction may well base decisions on the distance and speed of approaching vehicles, and have a time dimension in that as they spend longer at the T-junction, they are prepared to risk a smaller the gap for the

approaching vehicles due to pressure to pull out, and greater confidence in their ability to pull out in time through experience of watching the traffic at the junction.

In terms of general applicability of the cumulative approach, the use of the OUTDATA approach is a limitation in that it can only be used for decisions with discrete outcomes. It is questionable whether a decision with a continuous output would offer the same problems as the renege decision since it would have much less of an all or nothing nature to it. Part of the problem with the renege is that once a customer had reneged, no further decisions were made, but the probabilities of reneging were conditional on them having stayed up until then. The continuous output decision is unlikely to follow that pattern, so that a cumulative distribution is less likely to be required. If a cumulative distribution was required, the INVAR approach could be used, and the effect on the size of the training data, and therefore training time, would have to be accepted. The idea of generating a single random number for a particular entity to generate more consistent behaviour could be used whether a cumulative probability was being used or not.

The renege decision model is limited in the number of decision criteria it uses, which makes the job of dealing with the decision groups easier. The sequencing tasks for the decision groups in implementing the reneging decision module was relatively straight forward. However, as the number of decision criteria dimensions increases, the task of dealing with the groups is likely to become increasingly complex in terms of training the network, but even more so for sequencing the changes between decision groups when the model is implemented. This is likely to be a key factor limiting the size of models that can be developed.

It is also worth considering the cases where the neural network models were not used. In the case of deciding which queue to join, the Transactions service offers most opportunity for discussion. Here, between 1 and 5 counters could be open with a number of different combinations of open and closed counters. The choice of which queue to join involves a high spatial element and so the specific patterns of counters available need to be included in a neural network. The number of possible patterns is

large, requiring a lot of data for the neural network to get reasonable picture of the patterns. The neural network should be able to do some interpolation between patterns but this is a more difficult task than interpolation between the values of a variable. Thus, unless very large amounts of data are available, the use of neural networks for spatial type problems would be doubtful. The use of the hybrid model would however allow any non-spatial sub-decisions to be modelled, as shown by the balk/stay decision and then a rule-based or abstraction approach used to deal with the spatial aspects.

The other decision that was not modelled was the queue swapping. This was mainly down to the use of the data collection framework and the nature of the decision. Queue swapping involves competition between customers, and this is difficult to capture in an artificial data collection framework where there is only one subject at a time. Had the appropriate data been available, it would still be likely to be a difficult situation to model. On an individual basis, the decision is likely to have a time dimension in terms of how long people would spend weighing up the decision before deciding to swap queues. If somebody did not swap it would be difficult to determine whether they did not want to, or whether they just spent too long thinking about it. It might be easier to consider the decision on a queue basis first, to determine which queue a customer would be swapping from, before going down to a decision at the individual level.

A number of specific issues have been raised by the bank simulation, and there has been some effort to identify these and determine how far they can be generalised. This leads to a consideration of the advantages and disadvantages of the approach in more general terms.

7.5 Strengths and Weaknesses of the Approach

The main strengths and weaknesses of the approach are summarised. These are compared with the features of the alternative approaches, particularly abstraction and rule-based which are the most frequently used in simulation.

7.5.1 Strengths

The following strengths of the stochastic neural network approach have been identified:

- *Allows the consideration of the variability in decisions for different individuals*

The use of the stochastic elements in the modelling approach creates a distribution of decision responses to particular decision criteria. This does not treat the decision making processes of all people as being the same. This is useful where there is a requirement to try and model actual rather than optimal or idealised decision making.

- *Allows the effects of decision criteria to be taken into account in the model*

The approach allows decision criteria to be included as parameters in the model along with the stochastic elements. The approach is effectively a multi-dimensional distribution for the decision making process.

- *Data driven approach*

The data driven approach has two advantages. One is that a detailed knowledge elicitation process is not required. This is particularly useful if there is limited access to the decision makers for model development. Secondly, there is no assumption of logical rules to govern behaviour, rather the behavioural aspects of the decision making can be captured. This is particularly important for everyday and generally non-cognitive behaviour.

- *Flexibility in modelling the form of decision outcome*

The approach allows the modelling of decisions with continuous or class variables. This provides the opportunity to model a variety of situations. The results of the study suggest that in most cases the INVAR model will give the best generalised model, being applicable to both types of decision outcome. Some problems with continuous time dimensions can be modelled.

- *Hybrid approach reduces complexity of decision making models and allows the use of mixed techniques*

The hybrid model allows decision processes to be reduced into several sub-decisions. This reduces the dimensionality of some of the decisions, reducing the amount of data required, and making the models more general. It also allows a variety of approaches, not just neural networks to be used. This means that parts of the decision making where neural networks cannot be used, either through lack of data or the nature of the problem, can be tackled using other approaches. The neural network can be involved in part of the model but does not need to be able to represent all of the decision making process.

- *The consistency of decisions by an individual entity can be maintained by re-using a random number*

The stochastic process represents variety in decision making. In some cases, consistency for an individual is required. A random number, which might reflect their attitude to some process, can be maintained for that customer. The decisions of the individual might change as the values of the decisions criteria change, but the attitude remains consistent.

- *The structure of the neural network models can be visualised*

The neural network can produce fast responses to input criteria, allowing the creation of a large number of points for visualising the response surface. More decision criteria makes the visualisation more difficult to achieve, although dynamic computer generated models could be used. The visualisation of the models helps with understanding the dynamics of the decision making, and in validating the models.

- *Operation of trained neural networks is fast*

Once trained, the neural networks can produce responses quickly. This is important in the efficient operation of the simulation models.

7.5.2 Weaknesses

The following weaknesses of the stochastic neural network approach have been identified:

- *Data intensive*

The creation of the empirical probabilities requires sufficient data for reasonably accurate estimates of these probabilities to be achieved. More dimensions to the decision criteria means a greater data requirement. A spatial aspect to the decision can lead to a large combinatorial problem leading to very high data requirements. Requiring access to sufficient amounts of data is seen as one of the main factors that could prevent the use of the technique for particular applications.

- *Validation Data Requirements*

Proper numerical validation of the decision models requires decision making data that has not been used for developing the model. The stochastic nature of the models means that a variety of responses can be achieved for particular values of the decision criteria depending on the random variable. For validation, the test data set must also contain a variety of outcomes for a particular set of decision criteria so that the accuracy of the distribution can be evaluated. This means that large amounts of testing data need to be collected, a problem if there is a general shortage of data.

- *Difficulties in representing data with temporal aspects*

Some data involving temporal aspects require the use of cumulative probability distributions. The construction of these has been shown to be possible, but the process becomes increasingly problematic as the number of dimensions increase.

- *Lack of explanation facilities*

The neural networks can show the inputs being used, and the output response. The dynamics of the decision surface can also be visualised. However, the neural network models cannot explain the causal relationships that lead to a particular decision being made. This is unlikely to be possible unless a cognitivist (and therefore non-behavioural) approach is taken.

- *Long network training times*

The iterative training process for the multilayer perceptrons is time consuming, and increases with the number of criteria and amount of data that is used. This is largely a technology based issue, both in terms of the neural network algorithms and the power of computers. However, a substantial reduction in training times has occurred during the course of this study, particularly due to the improvements in computer hardware, and is set to improve further in the future.

7.5.3 Comparison with Other Approaches

A number of alternative approaches were reviewed in Section 2.5 and a comparison of the approaches made in Section 2.8. That comparison was made on the basis of the features of the approaches, and in the case of the neural network approach was on the theoretical grounds of what it might be able to achieve. Most of the arguments put forward still hold, but some can be refined in light of the experiences from the study.

The main approaches currently used for representing decision making in simulations are abstraction and rule-based. The main arguments in favour of neural networks against these approaches are the ability to represent stochastic processes, which is not easily done in the rule-based approaches, and the higher level of detail that can be modelled than with abstraction. Fuzzy sets were also mentioned as an approach for representing decision making. Many of the points for the rule-based approaches hold for fuzzy logic. Some of the concepts of inexact reasoning might be adapted to allow stochastic elements. However, the development of the logic is a more difficult task than for the other rule-based approaches.

The data driven aspect of the neural network modelling was viewed as an advantage of the approach, but it has been found that large amounts of data are required to produce the models. The abstraction approach is also data driven, but requires less data due to its lower complexity. The rule-based approaches require knowledge elicitation, and require an understanding of the causal relationships between the decision criteria. The use of rule induction can make the rule-based approach data driven, however the

approach is limited to representing decisions with class output values. The rule-based approach in general, whether using rule induction or not tends to struggle with representing non-linear relationships.

The hybrid approach, splitting up the decision process into sub-decisions and using neural network and rule-based approaches, tries to capture some of the strengths of both approaches. The development of rules for decisions is a reductionist approach, producing simpler sub-models, and this aspect is to some degree captured by the hybrid approach. While not reducing the decisions to the level of the rule-based approach, it provides simpler, more general situations to model with the neural networks and so reduces the data requirements.

The decisions in the bank simulation involving which queue to join, and whether a customer would swap queues involved producing models with a mixture of rules, and fairly simple stochastic abstractions. In some respects these reflect the hybrid approach without the neural network component. In these cases, the rules were generally more complicated and the stochastic distributions less detailed than for the models including the neural networks. This is perhaps the best indication of the differences between the approaches. The neural networks took onboard some of the detailed causal relationships that the rules were trying to capture, and allowed more interaction between decision criteria to be captured than for the stochastic abstractions. However, in these two cases, the data requirements of the neural networks were too high for them to be used.

Validation of the models is more difficult for the neural networks than for the other approaches. Abstraction, being simpler, generally requires less data for validation, and the main stochastic parts can usually be visualised if required. The rule-based approaches also require less validation examples, although this is really down to the lack of stochastic elements. A drawback of the rule-based approaches is the difficulty in visualising the decision, which is the case even where there is an explanation facility available. The neural networks require large amounts of data for numerical validation, although they do allow visualisation of the decision response surface.

All of the approaches have difficulties in dealing with cases involving decisions over time. It could be said that the simulation as a whole is an example of how abstraction deals successfully with the time dimension. However, the behaviour of the components is highly structured, and it is the difficulty in dealing with the decision making components that led to this study in the first place. Rules for the renege decision could be developed although they would be less flexible than the neural network, exhibiting no variation between customers. It would be possible to stochastically generate cut off points at which the customers would decide to renege under certain conditions (perhaps a maximum average service time), although the question remains about what distribution would be used to generate these cut off points. Abstraction would be less able than the neural networks to take into account the changing circumstances in the queue. This would be likely to involve a simple cut off point sampled when the customer joined the queue.

Looking at model building effort, it is clear that the neural network approach does require some effort in data collection, preparing data, network training times and comparing the results of networks with different numbers of hidden nodes. Abstraction is likely to require less modelling effort, requiring less data collection and simple model building. For the rule-based approaches the knowledge elicitation can be time consuming and difficult, particularly if there is little access to the decision makers. Alternatively, more arbitrary rules can be built up if only a rough representation of the decision making is required.

In the final analysis, the neural networks can be seen as having features that are not represented by the other approaches, but also difficulties in application. The stochastic neural network approach and hybrid approaches are an addition to the modeller's tool box whose pros and cons must be weighed up against the other approaches in deciding which to use. The choice really depends on how much the accuracy of the representation is seen to be important and needs to be weighed up against the effort in developing the models. The ease of data collection, and the amount of data available will also be key criteria. If optimal or idealised decision making is required, then a

largely rule based approach is probably more appropriate since variability would not be of interest.

7.6 Application in Practice

In order to use the stochastic neural network approach in practice a number of general conditions need to hold for it to be considered.

1. There is a wish to model actual decision making behaviour, with variety between individual decision makers, rather than idealised or optimal behaviour.
2. The decision making is considered important enough to the simulation results to warrant the effort of modelling it in detail.
3. It is possible to collect decision making data in sufficient quantities to generate the empirical probabilities for the neural networks for the desired number of decision criteria dimensions.
4. Decisions can be generalised to an extent that substantial quantities of data are not required to deal with a large number of very specific cases.
5. An explicit model of the causal relationships in the decision making is not a goal of the application in its own right, i.e. where the development of a rule-base is required for purposes other than the simulation model.

It is worth considering a number of applications in which the use of the stochastic neural network work approach might be considered. This is by no means definitive, but it does help to review some of the features of systems that might lead to the consideration of the approach. It must be remembered that each of the following areas involve some modelling issues that have not been considered in the Bank simulation. The stochastic neural network approach is not being suggested as a packaged solution. Some development would be required for each of the areas.

- *Queuing models*

The Bank simulation was an example of a case where customer queuing behaviour was modelled. Under the above conditions, it might be argued that in practice the use of

the stochastic neural network approach for modelling queuing behaviour in a bank would not warrant the effort that it would require. This is probably true, although in this study the Bank simulation was merely a vehicle for investigating the approach. However, it might be possible to build generalised queuing behaviour models for balking and reneging that could be used in a number of applications. Scaling the data to the same order of magnitude effectively normalises the relationships between the variables. Different queuing situations could be represented by using a single model, with the appropriate scaling being determined for the variables to reflect the specific situation. In this way, the network would represent the general pattern of behaviour, with the scaling used to relate this pattern to actual values. Thus several simulation models could benefit from the effort required to build a queuing decision model.

- *Vehicle driver behaviour*

An area of modelling where there would be interest in modelling actual behaviour is in the simulation of traffic systems where vehicles are being modelled on an individual basis. If models of driver behaviour can be developed, a comparison of different road layout and traffic control policies can be made via simulations. Typical decisions involving yes/no outcomes might be when to overtake another car, changing lanes on a dual carriageway, pulling out of a T-junction, and entering a roundabout system. Decisions with outcomes of a continuous nature would be changes in speed and following distance. Data collection for such models is an issue, and would require the use of fairly expensive equipment. Driving simulators could be attached to data collection equipment, or cameras could record behaviour that would then need to undergo computer analysis to measure the decision parameters.

- *Emergency evacuation models*

Behaviour of people in stress situations such as exiting a building in the event of a fire is often not rational and consistent for different individuals. Stochastic neural networks could perhaps be used to model decisions on the direction and speed of movement, and reaction to certain obstacles. One difficulty is the collection of data on behaviour, since this is not of prime concern to people in an emergency situation. Real-life simulations of emergency conditions can and have been carried out to

investigate behaviour. The use of infra-red cameras and other devices could be used to track movement and develop data for the decisions that are made. Such simulations are often costly events, and it would be unlikely to be done just to provide data for a neural network approach, however the data could be a by-product of such exercises.

- *Ecosystem simulations*

The behaviour of humans and other animals in an ecosystem could be represented by the stochastic neural network approach. Some discrete-event simulation models have been created for ecological studies such as that by Deadman & Gimblett (1994) for the behaviour of people in a forestry area. Decisions could involve the movement of individuals (or herds of animals) in an area in response to some stimuli such as other individuals, fire etc. There is no reason that the modelling of behaviour need be restricted to just humans, particularly with the problems of knowledge elicitation from animals.

- *Behaviour of units in military situations*

The simulation of behaviour in military situations is a complex issue, and it is not being suggested that neural networks can cope with this. There is likely to be considerable effort required to develop a rules to represent the causal relationships in decision making. However, areas might exist where modelling of behaviour needs to have a stochastic element because units may react differently to particular situations and coping with the results of this could be of interest. In these cases the use of stochastic neural networks could be considered.

- *Modelling behaviour to financial risk*

One possible application area is modelling how people behave with respect to financial risk. Martin & Moon (1992) and Moon & Martin (1996) analysed the decisions of people with respect to economic search, that is searching for the lowest price good when there are search costs involved. In both studies, simulations were used to collect data on peoples decision making, the results of which were used to develop a number of heuristics for behaviour. It was shown that different heuristics applied to different people. A stochastic neural network could be used to represent the variability. This

could be used in a simulation of behaviour, or purely to visualise the decision surface. Such approaches could possibly be extended to customer reactions to a new product or marketing initiatives, or for the responses of financial markets to certain events.

Chapter 8

Conclusions and Future Research

The previous chapter contained a summary of the research with a detailed discussion of the study and its main outcomes. This chapter highlights the main conclusions and suggests areas for future research.

8.1 Conclusions

The initial stages of the research identified a new approach for representing decision making in simulation, which takes account of variability in those decisions. The stochastic neural network approach is a data driven technique to capture actual decision making behaviour, and allows the representation of differences in decision making for individuals.

The principle of the approach is based round the idea of developing a distribution of decision making with the decision criteria as parameters to the distribution. A neural network is used to fit the distribution from examples of decision making behaviour and to generalise from the data so that responses can be generated for new cases. This allows decision making behaviour to be included as a component in the simulation.

Three alternative data structures were identified for representing the stochastic elements of the decision making from which the neural network would be trained. All of the approaches have the same sets of decision criteria but handle the probabilities differently.

The INVAR model has a random variable as one of its input parameters which is used much as in an ordinary inverse density function to determine the outcome. The INVAR model can be used for cases where the outcome is represented by continuous or classification variables.

The OUTDATA model is used only for cases where the output values are classes. The probability of an outcome being in a particular class is represented in the output data that is used as targets in training the neural network. This means that the data must be processed to determine those class probabilities. In the use of the trained network, a random number is generated and compared against the outcome probabilities for a particular set of circumstances so as to determine the actual decision that is made by an entity.

The OUTNET model can also only be used where the outputs are classification values. The neural network is trained with the same input parameters as the OUTDATA approach, but uses the observed decision outputs. The network is allowed to develop its own probabilities for outcomes in each class as a way of dealing with examples that have conflicting results. The trained network represents the probabilities of choosing a class in the outputs much as for the OUTDATA model.

The results of the analysis with artificial data and using a case study involving representing customer queuing decisions in a bank, show that the approaches allow an accurate representation of decision making provided sufficient data examples are available. The key data requirement is to have enough examples to generate reasonably accurate estimates for empirical probabilities in the model. The artificial data sets showed that for cases with continuous valued outputs, extremely accurate results could be achieved with very good estimates for the probabilities, but that the accuracy diminished with less data available for estimating the probabilities. For the data sets with classification outputs, the OUTDATA model produced the best results when very accurate probability values were available, but the INVAR model was better when the probabilities were estimated. Poor results were achieved when there was little data for determining the probabilities empirically. The Bank simulation case study re-enforced the choice of the INVAR model, and showed that it generally produced smoother models of the decision processes, and was less affected by aberrations in the data than the other methods.

The aim of the study was to assess the feasibility of the stochastic neural network approach. The approach has been shown to be feasible, at least within the limitations of the problems covered in the study. The main benefits and drawbacks of the approach are summarised below.

The stochastic neural network approach has the ability to represent the stochastic elements of decision making by different people while still taking account of the factors that affect those decisions. The approach deals best with one off decisions, but repeated decisions can be modelled by sampling and retaining a random number for a particular individual. This can be viewed as determining the attitude of the person which will then affect how they react to the specific decision situation. This can add to the complexity of the model in some cases where decisions become conditional on previous decisions.

The stochastic neural network approach is data driven, and so reduces the need for a knowledge elicitation stage, although some awareness of the key criteria in making the decisions is still required for building and validating the models. However, the approach has been shown to require large amounts of data for fitting the models due to the need to generate empirical probabilities for a number of combinations of the criteria values. More decision criteria mean a need for more data, and this is likely to be a limit to the number of criteria that can be modelled. The approach has particular problems when a number of very specific decision situations exist since these are difficult to generalise and require very large amounts of data.

Validation of the models is a problem. The approach has the useful feature that it allows the response surface of the decision distribution to be visualised through generating and graphing large numbers of network responses. However, numerical validation requires the use of decision making examples that have not been used in training and needs these in fairly large quantities. This creates difficulties where the availability of data is limited.

The use of a hybrid model adds some flexibility to the approach. This looks at ways of splitting the decision making process into simpler sub-processes. It has the aim of making the sub-processes more general and so reduces the data requirements for modelling them with the stochastic neural networks. Also it allows the use of non-neural network approaches, particularly rule-based ones, where lack of data or the features of the situation make the use of the neural network impractical for some parts of the decision making process.

In determining whether to use the approach in practice, a number of factors need to be considered. If these conditions hold, then use of the stochastic neural network approach is a possibility.

1. There is a wish to model actual decision making behaviour, with variety between individual decision makers, rather than idealised or optimal behaviour.
2. The decision making is considered important enough to the simulation results to warrant the effort of modelling it in detail.
3. It is possible to collect decision making data in sufficient quantities to generate the empirical probabilities for the neural networks for the desired number of decision criteria dimensions.
4. Decisions can be generalised to an extent that substantial quantities of data are not required to deal with a large number of very specific cases.
5. An explicit model of the causal relationships in the decision making is not a goal of the application in its own right, i.e. where the development of a rule-base is required for purposes other than the simulation model.

8.2 Future Research

The results have shown that the stochastic neural network approach is feasible, although there are some difficulties in its application. The experiments which have been undertaken represent a first step in testing out the approach to ascertain its feasibility. Areas of further work exist for the evaluation of the approach, development of a model building methodology and to enhance the technical aspects surrounding model building.

- *Further case studies to evaluate the approach*

The experiments with the artificial data and the Bank simulation provided cases for evaluating the stochastic neural network approach. The artificial data covered cases with continuous and discrete outputs, while the Bank simulation covered cases with discrete outputs and time related features. Other cases studies would involve looking at decisions with different background situations.

Of particular interest is a case that involves the practical application of models with continuous outputs, some of which have time related aspects. Also, all the models in the study involved cases with one or two decision parameters (plus the random number variable), so that an investigation of decisions with more decision criteria, and more data to be able to represent them, would be useful. In addition, most of the thought in this study has been about representations at an individual level. The possibility of representation at a group level, such as for a whole queue rather than individuals in the queue, or groups of animals, could be investigated.

- *Development of a model building methodology*

The study took a step by step approach to the development and implementation of the models. This was required since the work involved stepping into an area with little existing research and so the aim was to be as informed as possible at each step. Some issues were anticipated, such as the need to develop documentation to record the structure of the decision making modules and data files involved in creating the modules. Also standard prototyping techniques were used in implementing the models. However, with experience from further case studies it should be possible to learn lessons and develop problem structuring methods that would help smooth the model building process.

- *Enhancing the modelling techniques*

A number of modelling approaches for representing stochastic processes were tried. Further work should be done to refine the techniques so as to improve representational accuracy. It would be useful to investigate the effects of the balance between the

number of decision variable groups and the number of examples in each group. There may also be adaptations that can be made to the neural networks to improve the structure of the models, such as being able to produce generalised models without having to test networks with different numbers of neurons, and the development of monotonic networks for fitting cumulative probability distributions. There may also be non-neural network based fitting techniques that could be investigated as alternatives to the neural network approach.

The current work involved several pieces of software in the development of the models. Spreadsheet software and specialised Pascal programs were used to prepare, scale and visualise the data. General neural network software was used for model fitting, and then a reduced version used for inclusion in the Pascal-based simulation model. The development of software more specifically designed for the development of stochastic neural networks should help speed up the model development process.

- *Extension of approach to applications outside the discrete event simulation area*

Discrete-event simulation was the context for developing the stochastic neural network approach. Further possibilities could exist for representing multi-dimensional stochastic distributions that may or may not involve the modelling of decision making. The use of stochastic neural networks for the investigation of financial markets and financial decision making is one such area.

References

- ADELSBERGER H. H., POOCH U. W., SHANNON R. E. & WILLIAMS G. N. (1986) Rule based object oriented simulation systems. In P.A. Luker & H.H. Adelsberger (Eds.) *Intelligent Simulation Environments*, Proceedings of the Conference on Intelligent Simulation Environments, San Diego, California, January 1986, pp107-112. Society for Computer Simulation, San Diego : CA.
- ANDERSON J. & EVANS M. (1994) Intelligent agent modelling for natural resource management. *Mathl. Comput. Modelling*, 20:8, 109-119.
- ANDERSON J. A. (1972) A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14, 197-220.
- ANDERSON J. A. & ROSENFELD E. (1988) *Neurocomputing : Foundations of Research*. MIT Press, Cambridge : MA.
- ATHANASSOPOULOS A. D. & CURRAM S. P. (1996) A comparison of data envelopment analysis and artificial neural networks as tools for assessing the efficiency of decision making units. *J. Opl. Res. Soc.*, 47, 1000-1016.
- AZOFF E. M. (1994) *Neural Network Time Series Forecasting of Financial Markets*. Wiley, Chichester.
- BARRETT M. L. & BEEREL A. C. (1988) *Expert Systems in Business: A Practical Approach*. Ellis Horwood, Chichester.
- BIRTWISTLE G. M., DAHL O-J., MYHRHAUG B. & NYGAARD K. (1980) *Simula Begin*, 2nd Ed. Chartwell-Bratt, Bromley.
- BISHOP C. (1995) Training with noise is equivalent to Tikhonov regularization. *Neural Computation*, 7, 108-116.
- BODEN M. A. (1990) Introduction. In M. A. Boden (Ed.) *The Philosophy of Artificial Intelligence*, pp1-21. Oxford University Press, Oxford.
- BODEN M. A. (1994) New breakthroughs or dead-ends? *Phil. Trans. R. Soc. Lond. A*, 349, 1-13.
- BODEN M. A. (1995) AI's half-century. *AI Magazine*, 16:4, 96-99.
- BROOKS R. A. (1991) Intelligence without representation. *Artificial Intelligence*, 47, 139-159.
- BUCHANAN B. G., BARSTOW D., BECHTAL R., BENNETT J., CLANCEY W., KULIKOWSKI C., MITCHELL T. & WATERMAN D. A. (1993) Constructing an expert system. In F. Hayes-Roth, D.A. Waterman, D.B. Lenat (Eds.) *Building Expert Systems*, pp127-167. Addison-Wesley, Reading : MA.
- CHENG B. & TITTERINGTON D. M. (1994) Neural networks : A review from a statistical perspective. *Statistical Sciences*, 9, 2-54.
- CHURCH K. B. & CURRAM S. P. (1996) Forecasting consumers' expenditure: A comparison between econometric and neural network models. *Int. J. Forecasting*, 12, 255-267.

- CURRAM S. P. & MINGERS J. (1994) Neural networks, decision tree induction and discriminant analysis: an empirical comparison. *J. Opl. Res. Soc.* 45, 440-450.
- DEADMAN P. & GIMBLETT R. H. (1994) A role for goal-oriented autonomous agents in modelling people-environment interactions in forest recreation. *Mathl. Comput. Modelling*, 20:8, 121-133.
- DIETTERICH T.G. & MICHALSKI R.S. (1984) A comparative review of selected methods for learning from examples. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.) *Machine Learning : An artificial Intelligence Approach*, pp 41-81. Tioga Publishing, Palo Alto : CA.
- DREYFUS H. L., DREYFUS S. E. & ATHANASIOU T. (1986) *Mind over machine: The power of human intuition and expertise in the era of the computer*. Blackwell, Oxford.
- DTI (1993) *Neural Computing Applications Portfolio*. Department of Trade and Industry, London.
- DTI (1994a) *Best Practice Guidelines for Developing Neural Computing Applications*. Department of Trade and Industry, London.
- DTI (1994b) Radio Rentals Improves Customer Targeting. *Neural Edge*, 6, p5.
- DTI (1995a) Neurons forecast consumer demand in soft drinks industry. *Neural Edge*, 9, p1.
- DTI (1995b) Neural computing in finance. *Neural Edge*, 9, p4-5.
- DTI (1995c) Neural computers in retail. *Neural Edge*, 10, p4-5.
- DOUKIDIS G. I. (1987) An anthology of the homology of simulation with artificial intelligence. *J. Opl. Res. Soc.*, 38, 701-712.
- DUBOIS D. & PRADE H. (1980) *Fuzzy Sets and Systems : Theory and Applications*. Academic Press, New York.
- DUTTA S. (1993) *Knowledge Processing & Applied Artificial Intelligence*. Butterworth-Heinemann, Oxford.
- ENGLEMORE R. S., MORGAN A. J. & NII H. P. (1988) Introduction. In R.S. Englemore & A.J. Morgan (Eds.) *Blackboard Systems*. Addison-Wesley, Wokingham.
- FISHWICK P. A. (1988) The role of process abstraction in simulation. *IEEE Trans on Systems, Man & Cybernetics*, 18, 18-39.
- FLITMAN A. M. & HURRION R. D. (1987) Linking discrete-event simulation models with expert systems. *J. Opl. Res. Soc.* 38, 723-733.
- FLITMAN A. M. (1986) Towards the application of artificial intelligence techniques for discrete event simulation. *PhD Thesis*. University of Warwick, England.
- FORSYTH R. (1989) The expert system phenomenon. In R. Forsyth (Ed.) *Expert Systems: Principles and Case Studies*, 2nd Ed, pp3-21. Chapman and Hall Computing, London.
- GORR W. L., NAGIN D. & SZCZYPULA J. (1994) Comparative study of artificial neural network and statistical models for predicting student grade point averages. *Int. J. Forecasting*, 10, 17-34.

- HAGAN M. T., DEMUTH H. B. & BEALE M. (1995) *Neural Network Design*. PWS Publishing, Boston : MA.
- HARDGRAVE B. C., WILSON R. L. & WALSTROM K. A. (1994) Predicting graduate student success: A comparison of neural networks and traditional techniques. *Computers Ops. Res.*, 21, 249-263.
- HART A. (1989) Machine induction as a form of knowledge acquisition in knowledge engineering. In R. Forsyth (Ed.) *Machine Learning: Principles and Techniques*, pp 21-38. Chapman and Hall Computing, London.
- HART A. (1992) Using neural networks for classification tasks - Some experiments on datasets and practical advice. *J. Opl. Res. Soc.*, 43, 215-226.
- HAUGELAND J. (1981) The nature and plausibility of cognitivism. In J. Haugeland (Ed.) *Mind Design*, pp243-281. MIT Press, Cambridge : MA.
- HAWLEY D. D., JOHNSON J. D. & RAINA D. (1990) Artificial neural systems : a new tool for financial decision making. *Financial Analysts Journal*, 46:6, 63-72.
- HECHT-NEILSON R. (1990) *Neurocomputing*. Addison-Wesley, Wokingham.
- HINTON G. E., McCLELLAND J. L. & RUMELHART D. E. (1986) Distributed representations. In D.E. Rumelhart, J.L. McClelland & the PDP Research Group *Parallel Distributed Processing, Vol 1*. pp75-109. MIT Press, Cambridge : MA.
- HOPFIELD J. J. (1982) Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79, 2554-2558.
- HOPTROFF R.G. (1993) The principles and practice of time series forecasting and business modelling using neural nets. *Neural Comput & Applic*, 1, 59-66.
- HORNIK K. M. S. (1991) Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4, 251-257.
- HORNIK K. M. S. & WHITE H. (1989) Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 359-366.
- HURRION R. D. (1980) An interactive visual simulation system for industrial management. *Eur. J. Opl. Res.*, 5, 86-93.
- HURRION R. D. (1993) Representing and learning distributions with the aid of a neural network. *J. Opl. Res. Soc.*, 44, 1013-1023.
- JACKSON P. C. (1974) *Introduction to Artificial Intelligence*. Mason & Lipscomb, London.
- JACKSON P. C. (1990) *Introduction to Expert Systems*, 2nd Ed. Addison-Wesley, Wokingham.
- JOINES J. A., POWELL K. A. & ROBERTS S. D. (1992) Object-oriented modelling and simulation with C++. In J.J. Swain, D. Goldsman, R.C. Crain & J.R. Wilson (Eds.) *Proceedings of the 1992 Winter Simulation Conference*, Arlington Virginia, December 1992, pp145-153. Society for Computer Simulation, San Diego : CA.
- KAHNEMAN D., SLOVIC P. & TVERSKY A. (1982) *Judgement Under Uncertainty: Heuristics and Biases*. Cambridge University Press, Cambridge.

- KNAPP B. M., DUDLEY A. R. & RYDER J. S. (1987) Modelling techniques for simulation of submarine engagements. *J. Opl. Res. Soc.* 38, 891-898.
- KOHONEN T. (1972) Correlation matrix memories. *IEEE Transactions on Computers*, 21, 353-359.
- KOLODNER J. (1993) *Case-Based Reasoning*. Morgan Kaufmann, San Mateo: CA.
- LAW A. M. & KELTON W. D. (1991) *Simulation Modeling & Analysis*. McGraw-Hill, London.
- LIPPMANN R. P. (1987) An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4:2, 4-22.
- MAASOUMI E., KHOTANZAD A. & ABAYE A. (1994) Artificial neural networks for some macroeconomic series : A first report. *Econometric Reviews*, 13, 105-122.
- MARSH B. D., WILLIAMS T. M. & MATHESON G. L. (1990) The use of mixed Prolog/Fortran for battle simulation. *J. Opl. Res. Soc.* 48, 311-318.
- MARTIN A. & MOON P. (1992) Purchasing decisions, partial knowledge, and economic search. Experimental and simulation evidence. *J. Behavioral Decision Making*, 5, 253-266.
- MAYRAND É., LEFRANÇOIS P. & MONTREUIL B. (1993) Agent-oriented simulation model of manufacturing activities. *Integrat. Comp-Aided Eng.*, 1, 137-146.
- McCLELLAND J. L., RUMELHART D. E. & HINTON G. E. (1986) The appeal of parallel distributed processing. In D.E. Rumelhart, J.L. McClelland & the PDP Research Group *Parallel Distributed Processing, Vol 1*. pp3-44. MIT Press, Cambridge : MA.
- McCLELLAND J. L. & RUMELHART D. E. (1988) *Explorations in Parallel Distributed Processing : A Handbook of Models, Programs and Exercises*. MIT Press, Cambridge : MA.
- McCULLOCH W. S. & PITTS W. (1943) A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 115-133.
- MENTOR (1993) *Computer-aided learning tutor in OR : Simulation*, MENTOR Software, Strathclyde.
- MERKURYEVA G. V. & MERKURYEV Y. A. (1994) Knowledge based simulation systems - A review. *Simulation*, 62, 74-89.
- MINGERS J. (1986) Expert systems - experiments with rule induction. *J. Opl. Res. Soc.* 37, 1031-1037.
- MINGERS J. (1989) An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4, 227-243.
- MINSKY M. & PAPERT S. (1960) *Perceptrons*. MIT Press, Cambridge : MA.
- MOON P. & MARTIN A. (1996) The search for consistency in economic search. *J. Economic Behavior & Organisation*, 29, 311-321.

- MÜNDERMANN F. (1993) Fuzzy concepts for predicting the behaviour of other drivers on a highway. In E.D. Klement & W. Slany (Eds.) *Fuzzy Logic in Artificial Intelligence*, 8th Austrian Artificial Intelligence Conference, Linz, Austria, June 1993, pp 165-173. Springer-Verlag, London.
- NEWELL A. (1981) Physical symbol systems. In D.A. Norman (Ed.) *Perspectives on Cognitive Science*, pp37-85. Ablex, Norwood : NJ.
- NORMAN D. A (1981) What is cognitive science ?. In D.A. Norman (Ed.) *Perspectives on Cognitive Science*, pp1-11. Ablex, Norwood : NJ.
- O'KEEFE R. M. (1986) Simulation and expert systems - a taxonomy and some examples. *Simulation*, 46, 10-16.
- O'KEEFE R. M. & ROACH J. W. (1987) Artificial intelligence approaches to simulation. *J. Opl. Res. Soc.*, 38, 713-722.
- ÖREN T. I. (1994) Artificial intelligence in simulation. *Annals of Ops. Res.*, 53, 287-319.
- OZEL F. (1992) Simulation modeling of human behavior in buildings. *Simulation*, 58, 377-384.
- PADGETT M. L. & ROPPEL T. A. (1992) Neural networks and simulations: Modeling for applicatons. *Simulation*, 58, 295-305.
- PIDD M. (1992) *Computer Simulation in Management Science*, 3rd Edition. Wiley, Chichester.
- PIDD M. (1995) Object-orientation, discrete simulation and the three-phase approach. *J. Opl. Res. Soc.*, 46, 362-374.
- PINKOWSKI B. (1990) CLUSTERT - A simulation-based expert. *Simulation*, 54, 179-185.
- POLLOCK J. L. (1989) *How to Build a Person*. MIT Press, Cambridge : MA.
- QUINLAN J. R. (1983) Learning efficient classification features and their application to chess end games. In R.S Michalski, J.G. Carbonell & T.M. Mitchell (Eds.) *Machine Learning: An Artificial Intelligence Approach*, pp 463-482. Tioga Publishing, Palo Alto : CA.
- REDDY Y. V. R., FOX M. S., HUSAIN N., & McROBERTS M. (1986) The knowledge based simulation system. *IEEE Software*, 3:2, 26-37.
- REFENES A. N., AZEMA-BARAC M., CHEN L. & KAROUSSOS S. A. (1993) Currency exchange rate prediction and neural network design strategies. *Neural Comput. & Applic.*, 1, 46-58.
- RICHARD M. D. & LIPPMANN R. P. (1992) Neural network classifiers estimate Baysian *a posteriori* probabilities. *Neural Computation*, 4, 461-483.
- ROBERTSON P. (1986) A rule based expert simulation environment. In P. A. Luker & H. H. Adelsberger (Eds), *Intelligent Simulation Environments*, Proceedings of the Conference on Intelligent Simulation Environments, San Diego, California, January 1986, pp9-15. Society for Computer Simulation, San Diego : CA.
- ROSENBLATT F. (1958) The perceptron: a probabilistic model for information storage and organisation in the brain. *Psychological Review*, 65, 386-408.

- RUIZ-MIER S. & TALAVAGE J. (1987) A hybrid paradigm for modelling of complex systems. *Simulation*, 48, 135-141.
- RUMELHART D. E., HINTON G. E. & WILLIAMS R. J. (1986) Learning internal representations by error propagation. In D.E. Rumelhart, J.L. McClelland & the PDP Research Group *Parallel Distributed Processing, Vol 1*. pp318-362. MIT Press, Cambridge : MA.
- RUMELHART D. E. & McCLELLAND (1986) PDP models and general issues in cognitive science. In D.E. Rumelhart, J.L. McClelland & the PDP Research Group *Parallel Distributed Processing, Vol 1*. pp110-146. MIT Press, Cambridge : MA.
- RUMELHART D. E., WIDROW B. & LEHR M. A. (1994) The basic ideas in neural networks. *Comm. of ACM*, 37, 87-92.
- SCHMIDT J. W. & TAYLOR R. E. (1970) *Simulation and Analysis of Industrial Systems*. Irwin, Homewood : ILL.
- SENJOWSKI T. J. & ROSENBERG C. R. (1987) Parallel networks that learn to pronounce english. *Complex Systems*, 1, 145-168.
- SHANNON R. E., MAYER R. & ADELSBERGER H. H. (1985) Expert systems and simulation. *Simulation*, 44, 275-284.
- SHARDA R. (1994) Neural networks for the MS/OR analyst : An application bibliography. *Interfaces*, 24, 116-130.
- SHEIL B. (1987) Thinking about artificial intelligence. *Harvard Business Review*, 87:4, 91-97.
- SIETSMA J. & DOW R. F. J. (1991) Creating artificial neural networks that generalize. *Neural Networks*, 4, 67-79.
- SIMON H. A. (1995) Artificial Intelligence: An empirical science. *Artificial Intelligence*, 77, 95-127.
- SMITH M. (1993) *Neural Networks for Statistical Modeling*. Van Nostrand Reinhold, New York.
- SMITHSON M. (1987) *Fuzzy Set Analysis for Behavioral and Social Sciences*. Springer-Verlag, New York.
- STEFIK M., AIKINS J., BALZAR R., BENOIT J., BIRNBAUM L., HAYES-ROTH F. & SACERDOTI E. (1983) The architecture of expert systems. In F. Hayes-Roth, D.A. Waterman & D.B. Lenat (Eds) *Building Expert Systems*, pp89-126. Addison-Wesley, Reading:MA.
- STERN E. & SINUANY-STERN Z. (1989) A behavioural-based simulation model for urban evacuation. *Papers of Regional Science Assoc.*, 66, 87-103.
- SWINGLER K. (1996) *Applying Neural Networks : A Practical Guide*. Academic Press, London.
- TAMBE M., JOHNSON W. L., JONES R. M. , KOSS F., LAIRD J. E., ROSENBLOOM P. S. & SCHWAMB K. (1995) Intelligent agents for interactive simulation environments. *AI Magazine*, 16:1, 15-39.
- THESEN A. & TRAVIS L. E. (1992) *Simulation for Decision Making*. West Publishing, St. Paul:MN.

- THOMPSON R. F. (1993) *The Brain : A Neuroscience Primer*, 2nd Ed. Freeman, New York.
- TRIPPI R. R. & TURBAN E. (1990) Auto-learning approaches for building expert systems. *Computers Opns Res*, 17, 553-560.
- TURBAN E. (1992) *Expert Systems and Applied Artificial Intelligence*. Macmillan, New York.
- VARELA F. J. (1992) Whence perceptual meaning ? A cartography of current ideas. In F.J. Varela & J-P Dupuy (Eds) *Understanding Origins*, pp235-236. Kluwer Academic Publishers, Netherlands.
- VOGL T. P., MANGIS J. K. , RIGLER A. K., ZINK W. T. & ALKON D. L. (1988) Accelerating the convergence of the back-propagation method. *Biol. Cyb.*, 59, 257-263.
- WANG S. (1994) Neural network techniques for monotonic nonlinear models. *Computers Ops. Res.*, 21, 143-154.
- WARNSHUIS P. (1967) Simulation of two-way traffic on an isolated two-lane road. *Transpn. Res.*, 1, 75-83.
- WEINROTH J. & MADEY G. (1990) The interface between simulation and neurocomputing. In W. Webster & J. Uttamsingh (Eds.) *AI and Simulation - Theories and Applications*, Proceedings of the SCS Eastern Multiconference, Nashville, Tennessee, April 1990, pp1-5. Society for Computer Simulation, San Diego, CA.
- WIDROW B. & HOFF M. E. (1960) Adaptive Switching Circuits. 1960 IRE WESCON Convention Record, Part 4, pp96-104. IRE, New York.
- WIDROW B., RUMELHART D. E. & LEHR M. A. (1994) Neural networks : applications in industry, business and science. *Comm. of ACM*, 37, 93-105.
- WILDBERGER A.M. (1990) Neural networks as a modelling tool. In W. Webster & J. Uttamsingh (Eds) *AI and Simulation - Theories and Applications*, Proceedings of the SCS Eastern Multiconference, Nashville, Tennessee, April 1990, pp65-74. Society for Computer Simulation, San Diego : California.
- WILLIAMS T. M. (1996) Simulating the man-in-the-loop. *OR Insight*, 9:4, 17-21.
- WILLIAMS T. M., GITTINS R. P. & BURKE D. M. (1989) Replenishment at sea. *J. Opl. Res. Soc.*, 40, 881-887.
- WILLSHAW D. (1994) Non-symbolic approaches to artificial intelligence and the mind. *Phil. Trans. R. Soc. Lond. A*, 349, 87-102.
- WINOGRAD T. & FLORES C. F. (1986) *Understanding Computers and Cognition*. Ablex, Norwood : NJ.
- ZADEH L. A. (1973) Outline of a new approach to the analysis of complex systems and decision processes. *I.E.E.E. Trans on Systems, Man, and Cybernetics*, 3, 28-44.
- ZAHEDI F. (1991) An introduction to neural networks and a comparison with artificial intelligence and expert systems. *Interfaces*, 21:2, 25-38.
- ZEIGLER B. P. (1990) *Object-Oriented Simulation with Hierarchical, Modular Models : Intelligent Agents and Endomorphic Systems*. Academic Press, San Diego : CA.

Representing Intelligent Decision Making in Discrete Event Simulation: A Stochastic Neural Network Approach

Appendices

by

Stephen Curram

MSc (Warwick)

BSc (Kent)



University of Warwick

Warwick Business School

January 1997

Appendices

A : The Neural Network Software	A-1
A1 : Using the Neural Network Software	A-1
A2 : Neural Network Software Code	A-7
A2.1 Library Units	A-7
A2.2 Library Code Details	A-8
B : Bank Simulation Program	B-1
B1 : Program Structure	B-1
B2 : Input Distributions	B-2
B3 : Entity Attributes	B-3
B4 : MicroSim Commands	B-3
B5 : Bank Simulation Program Listing	B-23
C : Bank Simulation Scenarios	C-1
D : Bank Simulation - Participants Instructions	D-1
E : Example Decision Outcome Datasets	E-1
F : Bank Simulation Questionnaire	F-1
G : Decision Mapping Forms for Bank Simulation	G-1
H : Stay / Balk Decisions for the Bank Simulation	H-1
H1 : Information Service	H-1
H2 : Transactions Service	H-13
H3 : Business Service	H-24
H4 : Currency Service	H-28
I : Reneging Decision for the Bank Simulation	I-1
J : Implementation of Intelligence Modules in the Bank Simulation	J-1
J1 : Customer Attributes	J-1
J2 : Decision Module Code	J-2

Appendix A

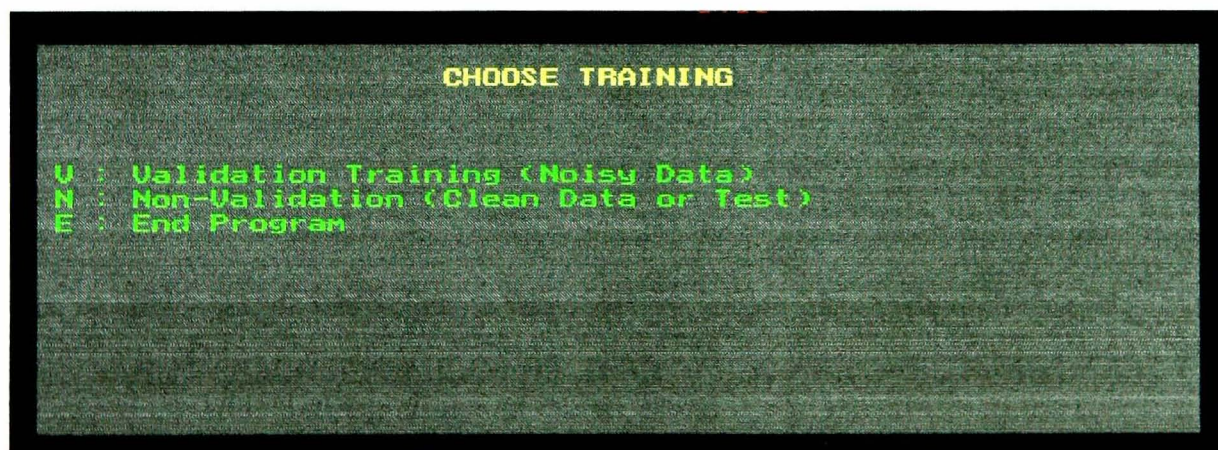
The Neural Network Software

A1 : Using the Neural Network Software

The neural network software was written using Borland Pascal 7 for DOS on an IBM PC compatible computer. It is designed using libraries of routines that hide most of the functionality of the neural network, allowing utilities to be called from a relatively simple main program.

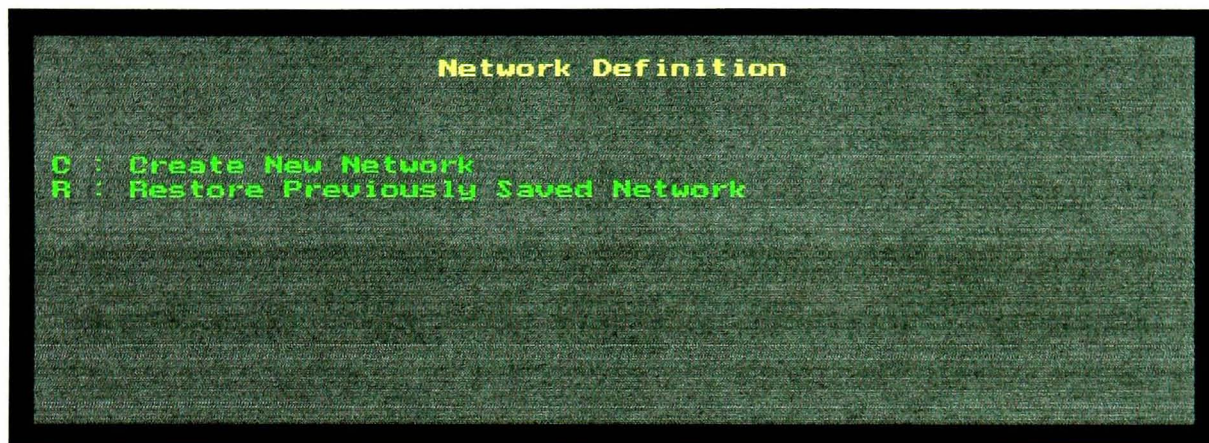
The software is designed to allow configuration of the network at run-time, and so does not require any subsequent programming to set up the network. It uses dynamic memory allocation via linked lists to create the network. The networks must have an input and output layer, and can have up to two hidden layers. The example data is stored in memory using dynamic arrays, with the amount of data being limited only by the amount of extended memory available on the PC.

The capabilities of the neural network software are best described by considering the sets of options available in setting up a network for training.



The first choice is the method of training to be used. **Validation training** involves splitting the example data into two parts, one part for training the network, that is determining the weight and node bias values, and the other part for validation which is used to determine the point during training that the network best generalises to new

data. This approach is usually used when the data is noisy. **Non-validation training** does not use a validation data set to determine when to stop training. It is usually used when the data does not contain noise, or when a trained network is being loaded so that it can be tested. In the experiments described in this thesis, the Non-validation approach was used because of the difficulty in setting up a validation data set, particularly with the lack of data available. The generalisation abilities of the networks were controlled by using different numbers of nodes in the hidden layer to set varying limits on the degree of non-linearity that could be modelled.

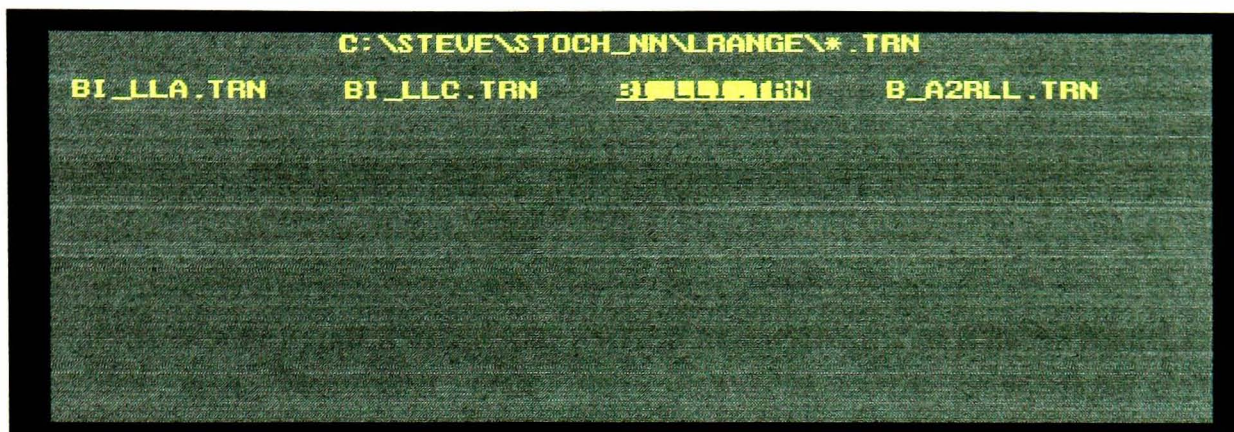


The next option is whether to create a new network, or load a previously trained network from file. Network training for a restored network is switched off, so that it can be used for testing (the training can be switched on again if desired). If a new network is being created, the number of nodes in each layer must be specified.



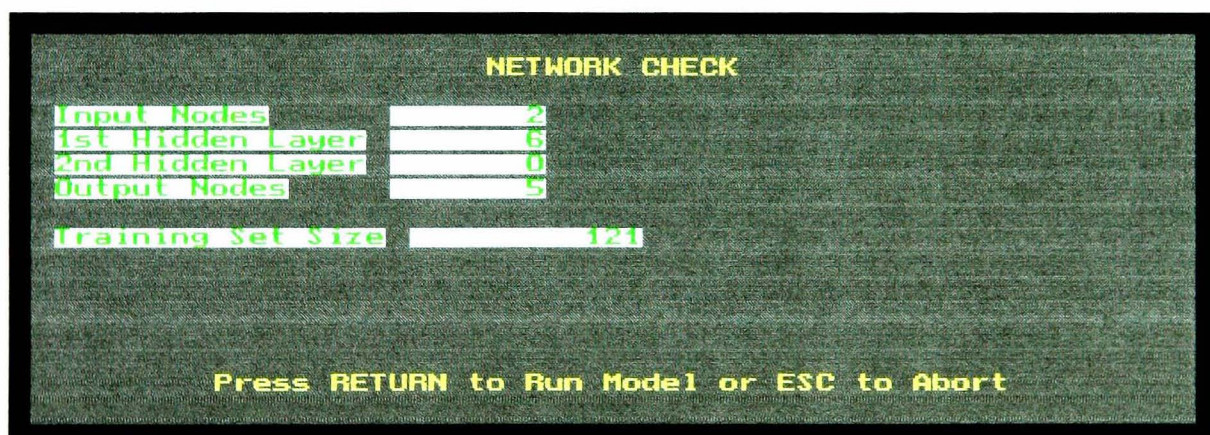
The network definition must have at least one node in each of the input and output layers. There can be zero, one or two hidden layers. In all of the experiments, only one hidden layer was used.

Next data must be loaded in for training the network. The data filename can be typed in, or a standard filename search string entered. A data file can then be selected from the set of matching filenames.



The data file is usually expected to be organised so that each example is specified in a single row, with the output target values at the end of each row. This is checked, and if the amount of data only matches the number of input values, a further file is requested containing the target values.

If the amount of data does not match the network specification then a warning is given and the network training cannot proceed, otherwise a confirmation of the network shape and number of examples in the data file is given.



The network training parameters can be specified at the start of training, and re-specified at any point during training.

Change Parameters	
A) Weight Gain	0.10
B) Weight Momentum	0.90
C) Bias Gain	0.10
D) Bias Momentum	0.90
E) Temperature	1.00
F) Weight Update Interval	121
G) Node Save Filename	node.nod
H) Weight Save Filename	weight.wgt
I) Save Interval	0
J) Actvn negligible level	0.50
K) Actvn medium level	1.00
L) Node plot->	AVERAGE VALUE
Z) Exit Menu	

Option A) allows the weight gain term to be specified. This controls the size of the steps used in updating the weight values during training. The larger the gain term, the larger the step size. A large gain term speeds up training, but it can allow the network to miss critical features on the error surface and so reduces the chance of finding a global minimum or near minimum.

Option B) allows the momentum term to be specified. The momentum represents an exponentially smoothed average of past weight changes. The term specifies the proportion of the old weight change value that will be used in the new one.

Options C) and D) specify the gain and momentum terms for the node biases. These are usually set to the same values as the weight terms.

The neural network software uses automatic parameter changing during training. The parameters are initially set with the gain terms at 0.1 and momentum terms at 0.9 (these are conventional figures to use). When the training error is reduced after an iteration, the gain terms are increased by 5%. If the training error gets worse by more than 2.5%, the last set of weight and bias updates are thrown away, the gain terms are reduced by 30%, and the momentum term is temporarily reduced to 0 until the training error starts to decrease again.

Option E) is a temperature term that effects the shape of the sigmoid function used in each of the nodes. A large temperature value ($T > 1$) pushes output values towards the extremes of 0 and 1, making the function more like a threshold function, while a small temperature value ($0 < T < 1$) makes the function more linear so that it acts like a hard

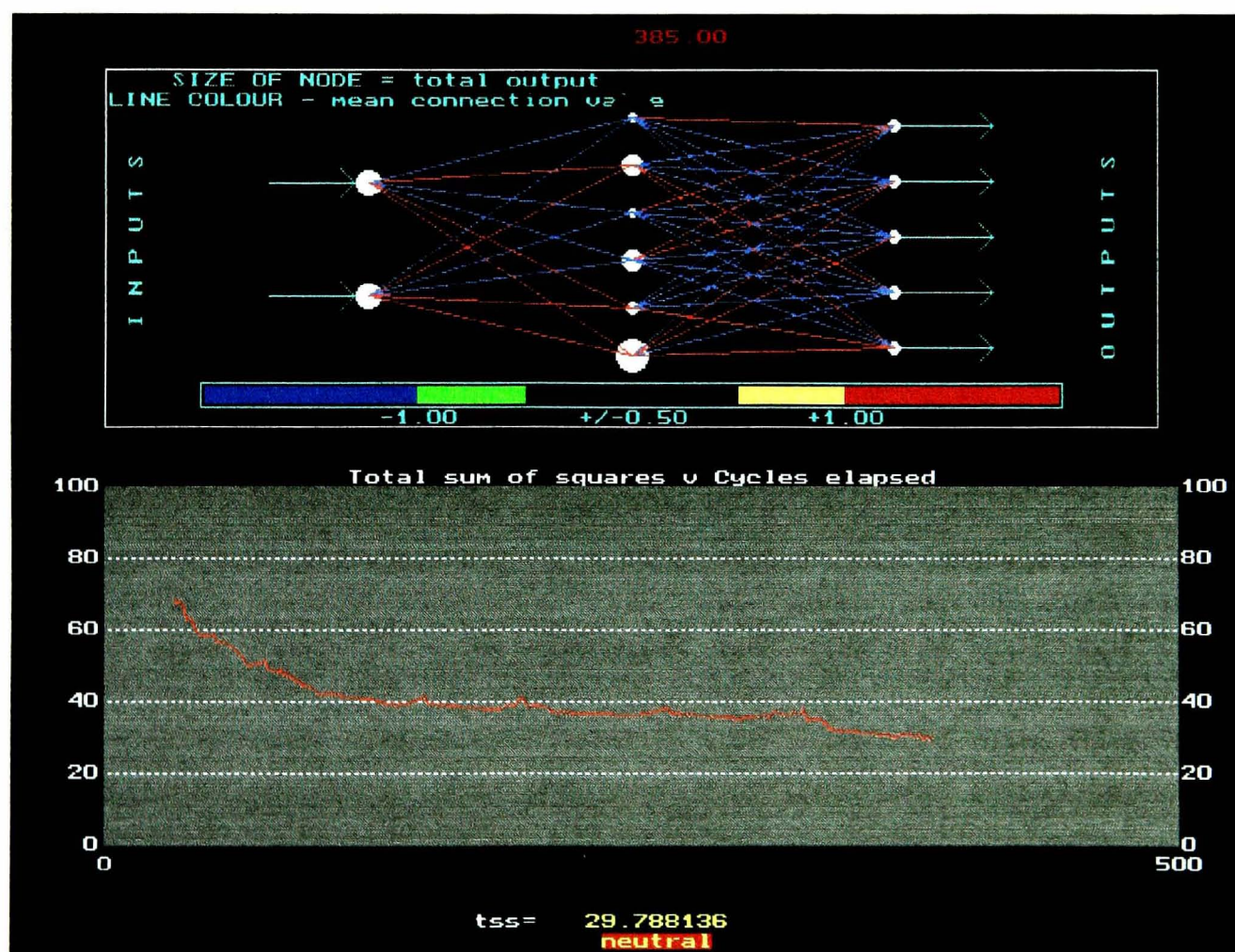
limiter. It is conventional to keep the temperature at 1 for an ordinary sigmoid function.

Option F) specifies the number of data examples to be passed through the network before the weight and bias terms are updated. Training is usually most efficient when updating occurs only after all of the examples have been passed through the network.

Options G) and H) allow the filenames to be specified for saving the network state.

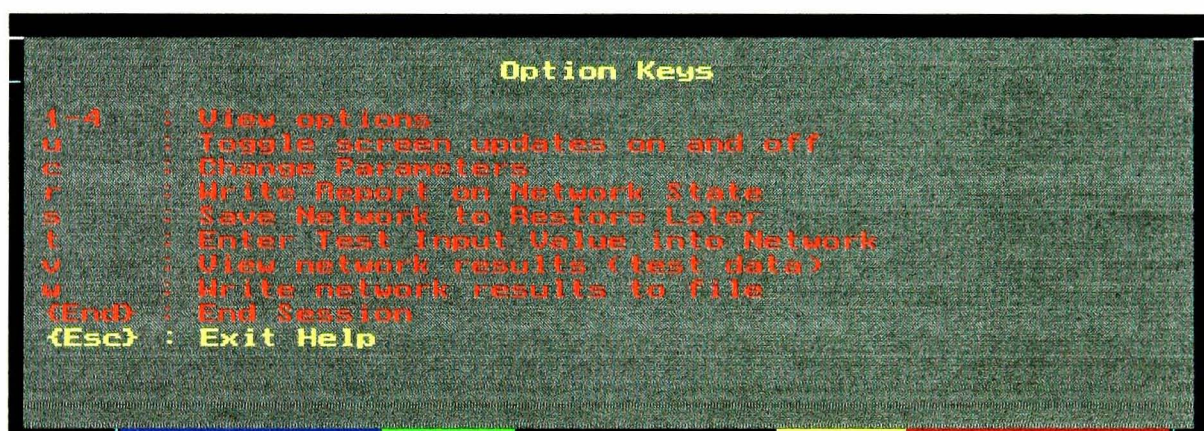
Option I) allows the network to be automatically saved after a specified number of iterations.

Options J), K) and L) are used to determine the features of the graphical display of the network shape, and are not central to training.



While the network is being trained, a number of alternative displays can be shown. The current iteration number is always displayed at the top of the screen, while the Total Sum of Squared errors is shown at the bottom. Also the state of training is displayed in terms of Success (error decreased), Neutral (error increased but by less than 2.5%) and Failure (error increased by more than 2.5%). Usually no other information is displayed since training runs faster that way. However, it is possible display the network node and weight activations for the last data example, show a graph of the training error, or show the state of the network diagrammatically. Each of the displays takes time to update, slowing down the training process, and so they are usually switched off.

A number of option keys exist to perform actions during network training. Pressing the appropriate keyboard key performs the action, or they can be accessed via a help screen (this is accessed by pressing **H** or **F1**).



Keys **1-4** allow different viewing options for the state of the network. Updating of the view options can be switched on and off using **U**. Pressing **C** (or **F2**) gives the Change Parameters menu that was first shown at the start of training. Key **R** writes a report on the network state, showing the number of iterations, parameter values and the weight and node bias values. **S** saves the state of the network under the filenames specified in Change Parameters. Pressing **T** pauses the network training and allows input values to be typed in so that the response of the network can be viewed. **V** allows the state of the network to be viewed for any of the training data, showing the training data values and network responses. **W** writes the network responses to a file, and is usually used when test data has been loaded into a trained network, so that the

network results can be analysed. The <End> key stops the network training. At this point a number of options are given for saving the network state or continuing with training.

Note that during the training of the networks for the experiments, conditions for stopping training were specified as a point after a given number of iterations (usually 50,000) where the minimum training error during the run was achieved. These conditions were built into a special version of the neural network main program, rather than being built into one of the unit library routines.

A2 : Neural Network Software Code

The code for the neural network software is divided up into a number of Pascal library units. Programs can then be written that utilise one or more of the units to create a neural network application. One main neural network application has been written that is used for the majority of work with the neural networks, although the experiments described in the thesis used this program with some minor modifications.

In the description of the coding, a general overview of each of the library units will be given, detailed descriptions and code will be shown for some of the main routines, and the neural network utility will be shown. Other programs created to help with the neural network training process will be described briefly.

A2.1 Library Units

The main neural network code is split between four library units called *Netkit*, *Valkit*, *NetIO* and *Simnet*.

Netkit contains the code necessary for setting up and using a neural network. This involves specifying the network shape, loading in previously saved networks, loading in data files, training the network (without validation) and using all of the network options.

Valkit adds the ability to train the network using a validation data set to determine when training should be stopped. Valkit contains some specially adapted code, and uses some Netkit code. Any program that uses Valkit would normally use the Netkit unit too.

NetIO contains a number of utilities for displaying graphics and text. Netkit and Valkit both contain default user interfaces that can be used, however NetIO can be used by application programs to help set up alternative user interfaces.

Simnet contains code to use trained neural networks in other programs. The code allows several neural networks to be loaded into memory and used in other Pascal programs. The Simnet library routines do not allow network training to be performed.

A2.2 Library Code Details

The neural network libraries amount to approximately 5000 lines of code between them. Much of it is to do with the user interface and options not central to the neural network training. Details will be given here for the code that deals with creating the neural network structure, training data usage and the training process. These are all contained in the Netkit unit. The code used in the Simnet library will also be described, since this was used to implement the networks in the simulation program described in Chapter 6. Valkit was not used for any of the experiments and so will not be described.

Netkit - Data Structures

The main data structures specify the records and pointers for the neural network nodes and weights. These are created using linked lists, although the connections between records are more complicated than simple one-way connections. A more detailed description of the structures is given with the code for creating the network.

The array used to contain the training data is specified as a single dimension array containing one element. Pascal does not have a proper method for defining dynamic arrays. However, an area of global memory can be specified and allocated to a pointer

for an existing array. The process involves overflowing the defined array bounds so that the {\$R+} Range Checking compiler directive can not be used, and the approach must be used with great care so as to prevent memory errors.

```

type
  outcometype = (success,neutral,failure);  { * training iteration outcomes* }

{ ** network node structure **}

  nodeptr = ^noderec;
  noderec = record
    activation:real;
    bias:real;
    old_bias:real;
    error:real;
    delta:real;
    bed:real;
    delta_bias:real;
    summation:real;
    min_value:real;
    max_value:real;
    next_node:nodeptr;
    prev_node:nodeptr
  end;

{ ** network weight structures **}

  weightleftptr = ^weightleftrec;
  weightrightptr = ^weightrightrec;

  weightleftrec = record
    value:real;
    old_value:real;
    wed:real;
    dval:real;
    next_to:weightleftptr;
    prev_to:weightleftptr;
    next_from:weightrightptr;
    node_from:nodeptr;
    summation:real;
  end;

  weightrightrec = record
    value:real;
    old_value:real;
    wed:real;
    dval:real;
    next_from:weightrightptr;
    node_from:nodeptr;
    summation:real;
  end;

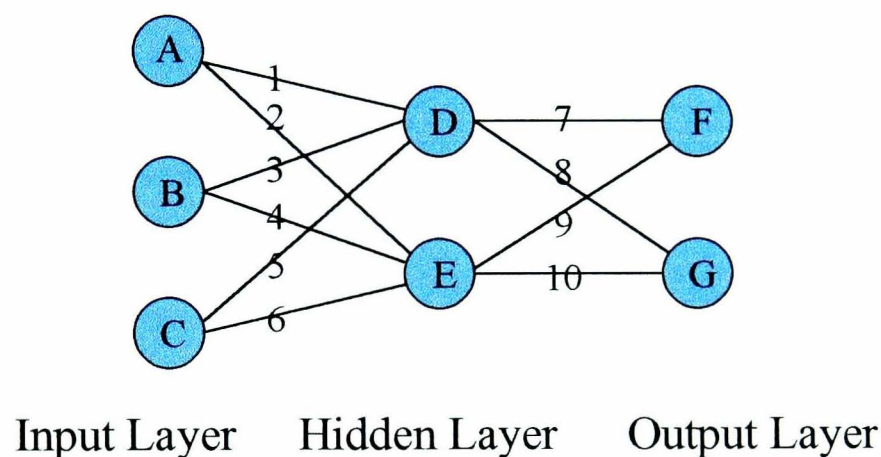
{ ** training data structure - memory allocated to array at run-time **}
{ ** when the required size of the array is known **}
  trainarray = array[1..1] of single;

```

Creating the Neural Network

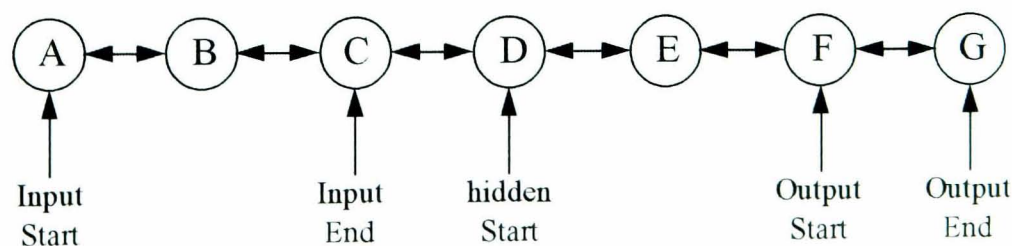
The neural network is constructed using sets of linked lists. The advantage of using the linked list approach is that memory can be dynamically allocated at run-time and so gives flexibility in specifying the network structures. The network training process generally involves an iterative movement through the weights and nodes. Sufficient pointer links have been added so that the linked lists can be processed during training without having to search for particular records.

The structures will be described with the use of an example network. In this case, the network has 3 input nodes, a single hidden layer with 2 nodes, and an output layer with 2 nodes. Diagrammatically the network would be organised as :

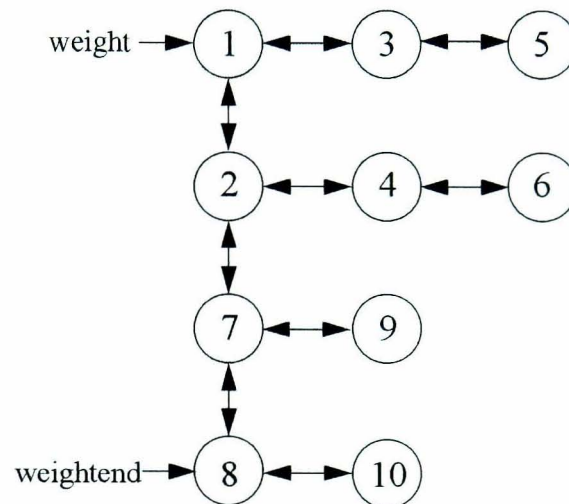


The network has 7 nodes with 10 weights between them.

The nodes are constructed as a single linked list with bi-directional links. Nodes from the input layer are at the start of the list, then nodes from the hidden layers are added, and finally nodes from the output layer are added. Extra pointers indicate the start and the end of the input layer, the start of each hidden layer, and the start and the end of the output layer.



The weights are organised into a structure similar to a grid. A linked list of 'left' weights represent each node that has a weight going to it, i.e. each node in the hidden and output layers. Each 'left' weight has a linked list of 'right' weights that represent all the weights coming from a node in the previous layer to that particular node. The pointer links are bi-directional, and pointers are used to indicate the first and last weights in the structure so that the grid can be worked through in both directions.



Each weight pointer also has a pointer to the record of the node from which it comes.

The process to create the network is controlled by the procedure *setup_nodes*. This first of all destroys any existing networks, creates the node linked list and then creates the weight linked lists.

```

{** network destruction routines **}

procedure destroy_nodes;
{** takes apart the network of node pointers & releases the memory used **}
var
  n:nodeptr;
  p:nodeptr;
begin
  n:=instart;
  instart:=nil;
  while n<>nil do
    begin
      p:=n;
      n:=n^.next_node;
      dispose(p)
    end
  end;
end;

```

```

procedure destroy_weights;
{** takes apart the network of weight pointers & releases the memory used **}
var
  wl:weightleftptr;
  pl:weightleftptr;
  wr:weightrightptr;
  pr:weightrightptr;
begin
  wl:=weight;
  weight:=nil;
  weightend:=nil;
  while wl<>nil do
    begin
      pl:=wl;
      wr:=wl^.next_from;
      while wr<>nil do
        begin
          pr:=wr;
          wr:=wr^.next_from;
          dispose(pr)
        end;
      wl:=wl^.next_to;
      dispose(pl)
    end
  end;
end;

procedure destroy_network;
{** calls routines to destroy a neural network if one exists **}
begin
  if instart<>nil then
    begin
      if training_size<>0 then
        undefine_training_set(training_size);
      destroy_nodes;
      destroy_weights
    end
  end;
end;

{** network creation routines **}

procedure build_weight_from(size:integer;var pl:weightleftptr;node:nodeptr);
{** attaches linked list to the left hand list to form a 2d dynamic 'array' **}
{** called once for every node that a weight goes to, creating weights for all **}
{** nodes that a weight comes from, to that particular node **}

var
  pr,lastr:weightrightptr;
  i:integer;

begin
  lastr:=nil;
  pl^.node_from:=node;  {** point to corresponding node **}
  node:=node^.next_node; {** this is done even if there are no other weights to be attached to the rhs weights **}
  for i:= 1 to size-1 do
    begin
      new(pr);    {** add a new weight node **}

```

```

if i=1 then
  pl^.next_from:=pr;

with pr^ do
  begin
    value:=-0.15+rnd(rn)*0.3;    {** initialize wieght **}
    old_value:=value;
    wed:=0.0;
    dval:=0.0;
    next_from:=nil;
    node_from:=node    {** point to corresponding node **}
  end;
  if lastr<>nil then
    lastr^.next_from:=pr;    {** add to linked list of weights **}
    lastr:=pr;    {** this record now last in list **}
    node:=node^.next_node
  end;
  pl:=pl^.next_to
end; {build_weight_to}

procedure create_weights;
{** creates the left hand side linked list for weights between nodes. the
  right hand side linked lists are then attached to this linked list **}

var
  i:integer;
  pl,lastl:weightleftptr;

begin
  lastl:=nil;
  weight:=nil;

  {**set up list of left hand wieghts**}

  for i:= 1 to (hid1size+hid2size+outputsize) do {** all non-input nodes **}
    begin
      new(pl);
      if weight=nil then {** first in list **}
        weight:=pl;
      with pl^ do
        begin
          value:=-0.15+rnd(rn)*0.3;    {** small random value **}
          old_value:=value;
          wed:=0.0;
          dval:=0.0;
          next_to:=nil;    {** pointer to next weight **}
          prev_to:=lastl;  {** pointer to last weight **}
          next_from:=nil;  {** next_right pointer **}
        end;
      if lastl<>nil then
        lastl^.next_to:=pl;    {** set pointer for last weight **}
      lastl:=pl
    end;
    weightend:=lastl;    {** last weight in linked list **}

  {** set up the right hand side linked lists **}

```



```

pl:=weight;

for i:=1 to hid1size do    {** weights from 1st hidden layer **}
  build_weight_from(inputsize,pl,instart);

for i:=1 to hid2size do    {** weights from 2nd hidden layer **}
  if hid1size>0 then {** two hidden layers **}
    build_weight_from(hid1size,pl,h1start)
  else {** one hidden layer (if defined as 2nd)**}
    build_weight_from(inputsize,pl,instart);

for i:=1 to outputsize do {** weights from output layer **}
  if hid2size>0 then {** two hidden layers **}
    build_weight_from(hid2size,pl,h2start)
  else if hid1size>0 then {** one hidden layer **}
    build_weight_from(hid1size,pl,h1start)
  else
    build_weight_from(inputsize,pl,instart);
end;

procedure create_nodes;
{** creates a linked list of node records incorporating all levels of the
neural network **}

var
  p,last:nodeptr;
  i:integer;

begin
  last:=nil;
  instart:=nil; {** start of input layer **}
  h1start:=nil; {** start of 1st hidden layer **}
  h2start:=nil; {** start of 2nd hidden layer **}
  outstart:=nil; {** start of output layer **}

  for i:= 1 to inputsize do
    begin
      new(p);
      if instart=nil then {** first input node **}
        instart:=p;
      with p^ do
        begin
          activation:=0;
          bias:=0;
          old_bias:=0;
          error:=0;
          delta:=0;
          bed:=0;
          delta_bias:=0;
          summation:=0.0;
          next_node:=nil;
          prev_node:=last
        end; {with}
      if last<>nil then
        last^.next_node:=p;
      last:=p
    end; {for}

```



```

inend:=last;          {** last node in input layer **}

for i:= 1 to hid1size do
begin
  new(p);
  if h1start=nil then  {** first node in 1st hidden layer **}
    h1start:=p;
  with p^ do
  begin
    activation:=0;
    bias:=-0.15+rnd(rn)*0.3;
    old_bias:=bias;
    error:=0;
    delta:=0;
    bed:=0;
    delta_bias:=0;
    summation:=0.0;
    next_node:=nil;
    prev_node:=last
  end; {with}
  if last<>nil then
    last^.next_node:=p;
  last:=p
end; {for}

for i:= 1 to hid2size do
begin
  new(p);
  if h2start=nil then  {** first node in 2nd hidden layer **}
    h2start:=p;
  with p^ do
  begin
    activation:=0;
    bias:=-0.15+rnd(rn)*0.3;
    old_bias:=bias;
    error:=0;
    delta:=0;
    bed:=0;
    delta_bias:=0;
    summation:=0.0;
    next_node:=nil;
    prev_node:=last
  end; {with}
  if last<>nil then
    last^.next_node:=p;
  last:=p
end; {for}

for i:= 1 to outputsize do
begin
  new(p);
  if outstart=nil then  {** first node in output layer **}
    outstart:=p;
  with p^ do
  begin
    activation:=0;
    bias:=-0.15+rnd(rn)*0.3;
    old_bias:=bias;

```

```

        error:=0;
        delta:=0;
        bed:=0;
        delta_bias:=0;
        summation:=0.0;
        next_node:=nil;
        prev_node:=last
    end; {with}
    if last<>nil then
        last^.next_node:=p;
        last:=p
    end; {for}
    outend:=last;          {** last node in output layer **}
end; {create nodes}

procedure setup_nodes(inp,h1,h2,outp:integer);
{** sets-up a neural network and weights with the specified number of
  nodes in each layer - setting h2 and/or h1 to 0 specifies 1 or 0
  hidden layers **}

begin
{** first destroy any existing networks (if any) **}

    destroy_network;

{** then create a new network provided there is at least one input node
  and one output node **}

    {** default parameter values **}
    if (inp<>0) and (outp<>0) then
        begin
            inputsize:=inp;
            hid1size:=h1;
            hid2size:=h2;
            outputsize:=outp;
            temperature:=1;
            momentum:=0.9;
            momentum_set:=0.9;
            gain:=0.1;
            bias_momentum:=0.9;
            bias_momentum_set:=0.9;
            bias_gain:=0.1;
            tss:=1;

            create_nodes;
            create_weights
        end
    end;
end;

```

Storing and Using the Training Data

The training data is stored in two dynamic arrays, one for the input data and one for the target data. Memory is allocated from the Global Heap, so that data storage is limited only by the amount of extended memory available to the PC. The approach

means that in effect only a one-dimensional array can be used. The training data storage requires two-dimensions (example number, variables) so conversion functions were created that allowed two-dimensional indexes to be specified, and converted these to a one-dimensional data storage format. Since Pascal does not officially allow dynamic arrays, a dummy array is created, the required amount of memory is allocated to it, and then the range of the dummy array is overflowed. For Pascal to allow this, the Range overflow compiler option must be switched off.

The following procedures create and destroy memory for data storage.

```

procedure define_training_set(train_size:longint);
{** defines the size of the dynamic arrays holding the training set data **}
{** Memory is allocated from the Global Heap to the inarray and outarray **}
{** pointers **}
{** memory is allocated for each data item, plus a spare row of data that **}
{** is used for temporary storage of results **}
var
  HI:THandle;
  HO:THandle;
begin
  training_size:=train_size;
  update_interval:=train_size;
  HI:=GlobalAlloc(gmem_Moveable,(train_size+1)*inputsize*sizeof(single));
  inarray:=GlobalLock(HI);
  HO:=GlobalAlloc(gmem_Moveable,(train_size+1)*outputsize*sizeof(single));
  outarray:=GlobalLock(HO);
end;

procedure undefine_training_set(train_size:longint);
{** free Global Heap memory attached to the inarray and outarray pointers **}
var
  HI:THandle;
  HO:THandle;
begin
  training_size:=0;
  HI:=GlobalFreePtr(inarray);
  HO:=GlobalFreePtr(outarray);
end;

```

The following procedures allow referencing of a two-dimensional array and convert it to one-dimensional format. *Getin* and *gettarget* allow data to be read from the arrays. *Putin* and *puttarget* allow data to be placed into the arrays. *Store_test_result* stores the network outputs in the 0th row of the target data array (reserved for temporary data storage).


```

function getin(i,j:longint):real;
{** get the i'th trial number input value for the j'th input node **}
{** the values are stored in a 1 dimensional array with dynamic size so
    this function is used to simulate a 2 dimensional form **}
begin
    getin:=inarray^[i*inputsize+j]
end;

function gettarget(i,j:longint):real;
{** get the i'th trial number target value for the j'th output node **}
{** the values are stored in a 1 dimensional array with dynamic size so
    this function is used to simulate a 2 dimensional form **}
begin
    gettarget:=outarray^[i*outputsize+j]
end;

procedure putin(i,j:longint;value:real);
{** add an i'th trial number value for the j'th input node **}
{** this is entered into the 1 dimensional dynamic (in terms of memory
    allocation - not a linked list) **}
begin
    inarray^[i*inputsize+j]:=value
end;

procedure puttarget(i,j:longint;value:real);
{** add an i'th trial number value for the j'th output node **}
{** this is entered into the 1 dimensional dynamic (in terms of memory
    allocation - not a linked list) **}
begin
    outarray^[i*outputsize+j]:=value
end;

procedure store_test_result;
{** storage of the activation values of the outputs in the 0th row **}
{** of the target data array - this is temporary storage only **}
var
    n:nodeptr;
    i:integer;
begin
    n:=outstart;
    for i:= 1 to outputsize do
        begin
            puttarget(0,i,n^.activation);
            n:=n^.next_node
        end
    end;

```

Neural Network Training

Training the neural network requires a series of operations to be done in sequence. For each data example, the network activations are calculated in response to the input

data, the output activations are compared to the target values, and then the errors are propagated back through the network to allocate them to weights and node biases, and the desired changes calculated. As more examples are presented to the network, the changes are summed (some will be positive and some negative). At some point, usually after the complete set of examples have been shown to the network, the values of the weights and biases are changed by the stored amount.

Node activations are calculated by working forwards through the network, while errors are attributed by working backwards (hence the multilayer perceptron is often known as a ‘feed-forward, back propagation network’).

The logistic function is used in all hidden and output nodes to generate activations in response to a value calculated as the sum of inputs to the node, plus the node bias.

```
function logistic_t(x:real):real;
{** the standard s shaped function for the forward feed - back propogation network **}
{** temperature changes the shape of the function
  - the higher the temperature - the steeper the curve of the s function
  - the lower the temperature - the flatter (and more spread out) the s
  - a temperature of 1 is the standard **}
{** x is the sum of inputs + the node bias **}
var
  xt:real;

begin
  xt:=-1.0*x/temperature;
  if xt>50.0 then
    logistic_t:=0.0
  else if xt<-50 then
    logistic_t:=1.0
  else
    logistic_t:=1.0/(1.0+exp(xt))
end;
```

The activations of all of the nodes are calculated for each of the data examples. This starts with the input nodes and works through to the outputs. The activations of the input nodes are simply the input data example values. The activations of each hidden and output node are the net of all of the weighted input values going into the node plus the node bias, with the net result being passed through the logistic function. From the first node in the hidden layer onwards, the weights can be matched up against the

linked list of left-weights, so that the weighted values from the nodes in the previous layer can be accessed.

```

procedure calc_output(t_no:longint);
{** calculate node activations for a particular trial **}
{** note that the sum of weighted inputs to a node is stored in the net variable **}
{** the summation, min and max values are used for the graphical display of the network **}
var
  i:longint;
  n:nodeptr;
  wl:weightleftptr;
  wr:weightrightptr;
  net:real;

begin
  n:=instart;
  {** start with input nodes - activation is merely the relevant input value **}
  for i:=1 to inputsize do
    begin
      n^.activation:=getin(t_no,i);
    { Maintain total of activation for each node over the cycle - for display }
    if t_no=1 then
      begin
        n^.summation:=n^.activation;
        n^.min_value:=n^.activation;
        n^.max_value:=n^.activation;
      end
    else
      begin
        n^.summation:=n^.summation+n^.activation;
        if n^.activation<n^.min_value then n^.min_value:=n^.activation;
        if n^.activation>n^.max_value then n^.max_value:=n^.activation;
      end;
      n:=n^.next_node;
    end;

    wl:=weight;
    {** for all other nodes in the network **}
    while n<> nil do
      begin
        {** calculate activations from nodes in previous layer multiplied by the **}
        {** relevant weight values and store in net **}
        net:=n^.bias;
        net:=net + wl^.node_from^.activation*wl^.value;
        if t_no=1 then
          wl^.summation:=wl^.node_from^.activation*wl^.value;
        else
          wl^.summation:=wl^.summation+wl^.node_from^.activation*wl^.value;
        wr:=wl^.next_from;
        while wr<>nil do
          begin
            net:=net + wr^.node_from^.activation*wr^.value;
            if t_no=1 then
              wr^.summation:=wr^.node_from^.activation*wr^.value;
            else

```

```

        wr^.summation:=wr^.summation+wr^.node_from^.activation*wr^.value;
        wr:=wr^.next_from;
    end;
    (** put net through the logistic function **)
    n^.activation:=logistic_t(net);

    { Maintain total of activation for each node over the cycle - for display }
    if t_no=1 then
    begin
        n^.summation:=n^.activation;
        n^.min_value:=n^.activation;
        n^.max_value:=n^.activation;
    end
    else
    begin
        n^.summation:=n^.summation+n^.activation;
        if n^.activation<n^.min_value then n^.min_value:=n^.activation;
        if n^.activation>n^.max_value then n^.max_value:=n^.activation;
    end;
    (** go onto next node, and next set of weights going into that node **)
    n:=n^.next_node;
    wl:=wl^.next_to
    end
end;

```

The network errors are calculated and attributed to individual nodes in the network. This is done backwards, starting with the outputs, and then attributing the errors to the hidden layer nodes using partial differentials.

```

procedure calc_error(t_no:longint);
(** calculates the error in the activations of the neural network when
    compared to the actual result **)
(** the errors are apportioned to individual nodes in the network **)
(** this is the back-propagation step **)
var
    n:nodeptr;
    wl:weightleftptr;
    wr:weightrightptr;
    i:longint;

begin
    (** initialise node errors to 0 **)
    n:=inend^.next_node;
    while n<>nil do
        with n^ do
            begin
                error:=0.0;
                n:=next_node
            end;
    end;

    (** output node errors are the difference between the target and activation **)
    n:=outstart;
    for i:=1 to outputsize do

```

```

with n^ do
  begin
    error:=gettarget(t_no,i) - activation;
    n:=next_node
  end;

{** working backwards, the errors are attributed to the other nodes **}
n:=outend;
wl:=weightend;
for i:=1 to outsize+hid1size+hid2size do
  begin
    with n^ do {** delta is differential of logistic function **}
      delta:=error*activation*(1.0-activation);
      wl^.node_from^.error:= wl^.node_from^.error + n^.delta * wl^.value;
      wr:=wl^.next_from;
      while wr<>nil do
        begin
          with wr^.node_from^ do
            error:=error + n^.delta * wr^.value;
            wr:=wr^.next_from
          end;
          n:=n^.prev_node;
          wl:=wl^.prev_to
        end
      end
    end;
  end;
end;

```

The desired weight changes for each weight and node are stored. These are summed over a number of examples.

```

procedure calc_wed(t_no:longint);
{** stores the accumulation of desired weight changes in the network **}
{** wed stores the sum of the weight changes to be made **}
{** bed stores the sum of the bias changes to be made **}
var
  n:nodeptr;
  wl:weightleftptr;
  wr:weightrightptr;

begin
{** calculate weight changes **}
  n:=inend^.next_node;
  wl:=weight;
  while n<>nil do
    begin
      wl^.wed:=wl^.wed + n^.delta * wl^.node_from^.activation;
      wr:=wl^.next_from;
      while wr<>nil do
        begin
          wr^.wed:=wr^.wed + n^.delta * wr^.node_from^.activation;
          wr:=wr^.next_from
        end;
      end;
{** calculate bias changes **}
      with n^ do
        bed:=bed + delta;
    end;
  end;

```



```

        n:=n^.next_node;
        wl:=wl^.next_to
    end
end;

```

The sum of square errors for the example pattern is calculated. Performing this operation for each of the examples gives the total sum of square errors.

```

function pss(t_no:longint):real;
{** calculate the pattern sum of squares **}
var
    n:nodeptr;
    v:real;
    i:longint;

begin
    v:=0.0;
    n:=outstart;
    for i:=1 to outputsize do
        begin
            v:=v + sqr(gettarget(t_no,i)-n^.activation);
            n:=n^.next_node
        end;
    pss:=v
end;

```

Changes to the weight and node bias values are sometimes made after each example is shown to the network, but usually it is after all of the examples have been shown. The previous procedures have calculated the sum of desired changes for each weight and node bias. These are applied, taking into account the gain (weight change step size) and the momentum (exponentially smoothed average of past changes).

```

procedure move_weights;
{** change the weights in the network according to the accumulated
    weight change values from each trial conducted between weight updates **}
{** storing the old values allows the changes to be undone **}
{** nodes and corresponding weights are accessed in parallel **}
var
    n:nodeptr;
    wl:weightleftptr;
    wr:weightrightptr;

begin
    {** update weight values **}
    n:=inend^.next_node;
    wl:=weight;
    while n<>nil do
        begin
            with wl^ do

```

```

begin
  old_value:=value;
  dval:=gain*wed + momentum*dval;  (** change in value **)
  value:=value + dval;  (** new value **)
  wed:=0.0  (** reset store of desired changes to 0 **)
end;
wr:=wl^.next_from;
while wr<>nil do
  with wr^ do
    begin
      old_value:=value;  (** store old weight value **)
      dval:=gain*wed + momentum*dval;  (** change in value **)
      value:=value + dval;  (** new value **)
      wed:=0.0;  (** change store of desired changes to 0 **)
      wr:=next_from
    end;
  with n^ do  (** update node biases **)
    begin
      old_bias:=bias;  (** store old bias value **)
      delta_bias:=bias_gain*bed + bias_momentum*delta_bias;  (** change in value **)
      bias:=bias+delta_bias;  (** new value **)
      bed:=0.0;  (** reset store of desired changes to 0 **)
      n:=next_node
    end;
  wl:=wl^.next_to
end
end;

```

Under some conditions the weight and bias changes need to be reversed. Since the old values were stored, this can be done.

```

procedure undo_update;
(** reverse previous weight and bias updates **)
var
  n:nodeptr;
  wl:weightleftptr;
  wr:weightrightptr;
begin
  n:=inend^.next_node;
  wl:=weight;
  while n<>nil do
    begin
      with wl^ do
        begin
          value:=old_value;
          wed:=0.0
        end;
      wr:=wl^.next_from;
      while wr<>nil do
        with wr^ do
          begin
            value:=old_value;
            wed:=0.0;
            wr:=next_from
          end;
        wr:=wr^.next_from;
      end;
    end;
  end;
end;

```

```

        end;
    with n^ do
    begin
        bias:=old_bias;
        bed:=0.0;
        n:=next_node
    end;
    wl:=wl^.next_to
end
end;

```

The training process is controlled by the *adapted_training_step* routine. This sequences the training steps for each data example, and applies the weight changes at the appropriate time. It also includes an adaptation that checks the status of training and allows the gain term to be changed.

```

procedure adapted_training_step;
{** sequences the training process - activates the network for each example **}
{** calculating the error and desired weight and bias changes **}
{** after the specified number of data examples have been processed, the **}
{** weight and bias changes are made **}
{** Also contains adaptive training where the bias term is changed according to **}
{** the success of the training step : **}
{** If the Total Sum of Squares (TSS) is reduced (success) the bias term is **}
{** is increased by 5% **}
{** If the TSS gets worse, but by less than 2.5% (neutral), everything is left **}
{** the same **}
{** If the TSS gets worse by more than 2.5% (failure) then the previous **}
{** weight changes are removed, the bias term is reduced by 30% and **}
{** the momentum term is reduced to 0 until the next successful step occurs **}

var
    i:longint;
    oldtss:real;

begin
    oldtss:=tss; {** previous total sum of squares **}
    tss:=0;
    display_cycle(cycle_no,300,10);
    for i:= 1 to training_size do {** amount of training data **}
    begin
        calc_output(i); {** activations **}
        calc_error(i); {** node errors **}
        calc_wed(i); {** desired changes **}
        tss:=tss+pss(i);
        if update_interval <>0 then {** weight update allowed **}
        if i mod update_interval = 0 then {** time to make weight update **}
        begin
            if tss/oldtss<1.0 then {** error reduced **}
            begin
                outcome:=success;
                gain:=gain*1.05;
                bias_gain:=bias_gain*1.05;
            end
        end
    end
end

```



```

        momentum:=momentum_set;
        bias_momentum:=bias_momentum_set;
        text_form(1,300,465,'success',0,1,yellow,red);
        move_weights
    end

    else if tss/oldtss<=1.025 then  {** small error increase **}
    begin
        outcome:=neutral;
        move_weights;
        text_form(1,300,465,'neutral',0,1,yellow,red)
    end
    else                          {** large error increase **}
    begin
        outcome:=failure;
        undo_update;  {** do not update weights, and undo previous change **}
        text_form(1,300,465,'failure',0,1,green,red);
        gain:=gain*0.7;
        bias_gain:=bias_gain*0.7;
        momentum:=0;
        bias_momentum:=0;
        tss:=oldtss;
    end;

    end;
end;
cycle_no:=cycle_no+1;
text_form(1,236,455,'tss= ',0,1,white,black);
longdec_form(1,268,455,tss,0,1,yellow,black);
if save_interval >0 then
    if round(cycle_no) mod save_interval = 0 then
        save_network
end;
end;

```

The code presented is the central code for setting up and using the neural network. The rest of the code involves loading in and saving data files, user options and data displays.

Neural Network Application

The neural network application program is effectively two programs combined. One uses ordinary network training, while the other one uses independent validation data to determine when to stop training.

The program contains code for settling a new network. Sometimes at the beginning of training, the network can move away to an extreme local minimum. This is caused by very large weight changes in the early training steps before the network has settled. Updating the network weights after each example has been shown prevents large

weight updates and so prevents the local minima. During the first stages of training, the network applies the weight changes after all of the examples have been shown to the network. If the error check is a failure (error increases) then weight changes are applied after each data example, for a complete iteration. This is repeated until a successful data step is achieved. After this the network will not move off to the extreme local minimum and so weight changes are only applied after a complete iteration.

```
{ $R- } { $N+, E+ }
program nnet;
{ ** neural network application, allows ordinary training ** }
{ ** or independent validation training ** }
uses
  crt, graph, netkit, netio, valkit;
var
  i, j: integer;
  c: char;
  special: boolean;
  show: boolean;
  view_option: integer;

procedure help(var special: boolean; var key: char);
{ ** this should be included in all programs - it contains instructions
  for the default option keys, and should be updated with those that
  the application program uses ** }
var
  ext: boolean;
  ch: char;
begin
  if (not special and (key='h')) or (special and (key=chr(59))) then
  begin
    open_own;
    text_form(1, 196, 10, 'Option Keys', 0, 1, yellow, darkgray);
    { ** default options ** }
    text_form(1, 8, 30, '1-4 : View options', 0, 1, lightred, darkgray);
    text_form(1, 8, 40, 'u : Toggle screen updates on and off', 0, 1, lightred, darkgray);
    text_form(1, 8, 50, 'c : Change Parameters', 0, 1, lightred, darkgray);
    { ** F2 also calls change_parameters ** }
    text_form(1, 8, 60, 'r : Write Report on Network State', 0, 1, lightred, darkgray);
    text_form(1, 8, 70, 's : Save Network to Restore Later', 0, 1, lightred, darkgray);
    text_form(1, 8, 80, 't : Enter Test Input Value into Network', 0, 1, lightred, darkgray);
    text_form(1, 8, 90, 'v : View network results (test data)', 0, 1, lightred, darkgray);
    text_form(1, 8, 100, 'w : Write network results to file', 0, 1, lightred, darkgray);
    text_form(1, 8, 110, '{End} : End Session', 0, 1, lightred, darkgray);
    { ** program options ** }
    text_form(1, 9, 120, '{Esc} : Exit Help', 0, 1, yellow, darkgray);

    repeat until hotkey(ext, ch);
    if ext or (ch <> chr(27)) then
      begin
```

```

        special:=ext;
        key:=ch
    end;
    close_own;
end
end;

{***** VALIDATION ROUTINES *****)}

procedure settle_new_val;
{** network may need learning steps to be reduced at beginning **}
{** to prevent convergence to a simple local minimum - this is **}
{** achieved by updating weights after each example. When the **}
{** adapted_training step succeeds, the network has settled **}
var
    network_settled:boolean;
begin
    network_settled:=false;
    while not network_settled do
        begin
            if outcome = failure then
                begin
                    update_interval:=1;
                    train_validate;
                    {** always gives outcome=success **}
                end
            else {** outcome neutral or success **}
                begin
                    update_interval:=training_size;
                    adapted_train_validate;
                    if outcome <> failure then {** after adapted **}
                        network_settled:=true
                    end;
                end;
            end; {** while not network settled **}
        end;

procedure netval;
begin
    node_savename:='default.nov';
    weight_savename:='default.wgv';

    repeat          {** program can be restarted from scratch **}
        user_setup; {** option of restoring or creating network **}
        data_setup; {** ask for datafilename & load data **}
        network_ok; {** check network, if ok then running:=true **}
        show:=true;

        if running then
            begin
                if newnetwork then
                    begin
                        change_parameters;
                        settle_new_val;
                    end
                end
            end
        end
    until not network_ok;

```



```

else
  begin
    update_interval:=0;
    training_step
  end
end;

{** if the network goes on for more than 2000 iterations without seeing a validation error **}
{** less than the observed minimum, then error very likely to have reached global minimum **}
{** so stop training - network state where global minimum was achieved will have been saved **}
while running and (worse_iterations<=2000) do
  begin
    {** put 1 cycle through the network **}
    adapted_train_validate;
    {** displays activations for last trial **}
    if show then
      case view_option of
        1: display_last_trial;
        2: plot_nodes(50,50,540,380,white);
        3: plot_mse(60,200,530,220,500,white,lightred,yellow,darkgray);
        4:begin
            plot_nodes(50,30,530,180,white);
            plot_mse(50,240,540,180,500,white,lightred,yellow,darkgray)
          end;
      end;
    end;

    {** test for key presses during the last cycle **}
    if hotkey(special,c) then
      begin
        help(special,c);
        if not default_keys(special,c) then
          case c of
            'a':test_auto_test;
            '1':view_option:=1;
            '2':view_option:=2;
            '3':view_option:=3;
            '4':view_option:=4;
            'u': begin
                  if show then
                    show:=false
                  else
                    show:=true
                  end
                end;
            if c in ['1','2','3','4'] then
              begin
                cleardevice;
                error_graph_visible:=false;
                node_display_visible:=false;
                node_coordinates_found:=false;
              end
            end
          end;
        end;
        if running then {** worse iterations caused stop **}
          finished:=true;
        cleardevice;
        if not finished then
          restart;

```

```

until finished;
end;

{***** NON-VALIDATION *****)}

procedure settle_new_network;
{** network may need learning steps to be reduced at beginning **}
{** to prevent convergence to a simple local minimum - this is **}
{** achieved by updating weights after each example. When the **}
{** adapted_training step succeeds, the network has settled **}

var
  network_settled:boolean;

begin
  network_settled:=false;
  while not network_settled do
    begin
      if outcome = failure then
        begin
          update_interval:=1;
          training_step;
          {** always gives outcome=success **}
        end
      else {** outcome neutral or success **}
        begin
          update_interval:=training_size;
          adapted_training_step;
          if outcome <> failure then {** after adapted **}
            network_settled:=true
          end;
        end;
      end; {** while not network settled **}
    end;

procedure netlearn;
begin
  repeat {** program can be restarted from scratch **}
    user_setup; {** option of restoring or creating network **}
    loaddata; {** ask for datafilename & load data **}
    network_ok; {** check network, if ok then running:=true **}
    show:=true;
    view_option:=0;
    if running then
      begin
        if newnetwork then
          begin
            change_parameters;
            settle_new_network;
          end
        else
          {** previously saved network - switch learning off to allow results **}
          {** to be recorded for original weights and biases **}
          begin
            update_interval:=0;
            training_step
          end;
        end;
      end;
    end;
  end;

```



```

while running do
  begin
    {** put 1 cycle through the network **}
    adapted_training_step;
    {** displays activations for last trial **}
    if show then
      case view_option of
        1: display_last_trial;
        2: plot_nodes(50,50,540,380,white);
        3: plot_tss(60,200,530,220,500,white,lightred,darkgray);
        4:begin
          plot_nodes(50,30,530,180,white);
          plot_tss(50,240,540,180,500,white,lightred,darkgray)
        end
      end;
    end;

    {** test for key presses during the last cycle **}
    if hotkey(special,c) then
      begin
        help(special,c);
        if not default_keys(special,c) then
          begin
            case c of
              '1':view_option:=1;
              '2':view_option:=2;
              '3':view_option:=3;
              '4':view_option:=4;
              'u': begin
                if show then
                  show:=false
                else
                  show:=true
                end
              end;
            if c in ['1','2','3','4'] then
              begin
                cleardevice;
                error_graph_visible:=false;
                node_display_visible:=false;
                node_coordinates_found:=false;
              end
            end
          end
        end;
        cleardevice;
        if not finished then
          restart
        until finished;
      end;

      {***** CHOOSE TRAINING *****}
procedure choose_training;
var
  ch:char;
begin
  repeat
    open_own:

```

```

text_form(1,168,10,'CHOOSE TRAINING',0,1,yellow,darkgray);
text_form(1,8,50,'V : Validation Training (Noisy Data)',0,1,green,darkgray);
text_form(1,8,60,'N : Non-Validation (Clean Data or Test)',0,1,green,darkgray);
text_form(1,8,70,'E : End Program',0,1,green,darkgray);
repeat
  ch:=readkey
until ch in ['V','v','N','n','E','e'];
close_own;
if (ch='V') or (ch='v') then
  netval
else if (ch='N') or (ch='n') then
  netlearn;
until (ch='E') or (ch='e');
end;

{***** MAIN *****)
begin
  setup;      {** setup system **}
  choose_training;
  closegraph;
end.

```

A slight adaptation was made to the program for training in the experiments, to stop the training automatically after a given number of iterations. Since the error can go up, and then come down again, training is not stopped until an iteration occurs where the error is the minimum observed. The relevant excerpt from the *netlearn* procedure is shown.

```

{auto stopping conditions}
  if (cycle_no >= 50000) and (tss < best) then
    begin
      save_network;
      running:=false;
      finished:=true;
    end;
  if tss < best then
    best:=tss;

```

Simnet Unit

The simnet unit contains code that allows trained neural networks to be used in other Pascal programs. Several neural networks can be loaded into memory and used at once. This is achieved by storing all of the relevant network parameters in a network record structure, so that several neural network variables can be created in single program.

The *set_network* function sets up a neural network, by loading in the network structure from the given files and allocating memory to the input data and output activations. Any loading errors while setting up the network cause the function to fail.

The *use_network* procedure activates the neural network, and causes it to store activation values in response to the inputs that it has been shown.

The *indata* procedure provides a value to one of the networks input nodes, while the *outdata* function retrieves a network output activation.

The use of the *simnet* routines can be seen in the description of the implementation of intelligence modules into a bank simulation in Appendix J.

```
{ $R- } { $N+ }
unit simnet;

interface

uses
  netkit;
type
  datarray=array[1..1] of real;

  {** nnet record allows local storage of values related **}
  {** to a particular network - this allows several neural **}
  {** networks to be used in the same program **}
  nnet = record
    firstnode:nodeptr;      {** first input node **}
    firstoutput:nodeptr;    {** first oupote node **}
    firstweight:weightleftptr; {** first weight record **}
    inp,h1,h2,out:integer;   {** layer sizes **}
    indata:^datarray;       {** input data array **}
    outdata:^datarray;      {** output data array **}
  end;

function set_network(nodename,wgtname:string; var network:nnet):boolean;
procedure use_network(var network:nnet);
procedure indata(network:nnet;index:integer;value:real);
function outdata(network:nnet;index:integer):real;

implementation
```



```

function set_network(nodename,wgtname:string; var network:nnet):boolean;
{** sets up a named neural network in memory. This network can only **}
{** be used for results, not for training. The definition of the **}
{** network is contained in node and weight files produced from an **}
{** application based on netkit **}

var
  fn,fw:text;
  e:integer;
  ok:boolean;
begin
  ok:=false;
  {$I-}
  assign(fn,nodename);
  assign(fw,wgtname);
  reset(fn);
  reset(fw);
  e:=IOResult;
  {$I+}
  if e = 0 then {** no error **}
  begin
    ok:=true;
    readln(fn); {** ignore cycle number in node file **}
    with network do
      readln(fn,inp,h1,h2,out); {** read network shape **}
    readln(fw); {** ignores cycle number in weight file **}
    readln(fw); {** ignores network shape in weight file **}

    {** the following pointers are used temporarily for the most **}
    {** recent neural network in memory. They are re-initialised **}
    {** for the new network **}
    instart:=nil;
    h1start:=nil;
    h2start:=nil;
    outstart:=nil;
    weight:=nil;

    with network do
      setup_nodes(inp,h1,h2,out); {** netkit routine to create net **}
    load_nodes(fn,e); {** loads node bias values into network **}
    load_weights(fw,e); {** loads weight values into network **}
    with network do
      begin
        {** store pointers to network in named network record **}
        firstnode:=instart;
        firstoutput:=outstart;
        firstweight:=weight;
        {** allocate memory for input and output data arrays **}
        getmem(indata,inp*sizeof(real));
        getmem(outdata,out*sizeof(real));
      end;
    end;
    set_network:=ok;
  end;

procedure use_network(var network:nnet);
{** passes input data through the specified network, and places results in the network

```

```

    record **}
var
  i:integer;
  n:nodeptr;
  wl:weightleftptr;
  wr:weightrightptr;
  net:real;
begin
  {** set current network pointers & size to specified network **}
  with network do
    begin
      instart:=firstnode;
      outstart:=firstoutput;
      weight:=firstweight;
      inputsize:=inp;
      hid1size:=h1;
      hid2size:=h2;
      outputsize:=out;
    end;

    n:=instart;
    {** load input data into network **}
    for i:=1 to inputsize do
      begin
        n^.activation:=network.indata^[i];
        n:=n^.next_node;
      end;

    wl:=weight;
    while n<> nil do
      begin
        net:=n^.bias;
        net:=net + wl^.node_from^.activation*wl^.value;
        wr:=wl^.next_from;
        while wr<>nil do
          begin
            net:=net + wr^.node_from^.activation*wr^.value;
            wr:=wr^.next_from;
          end;

        n^.activation:=logistic_t(net);
        n:=n^.next_node;
        wl:=wl^.next_to
      end;
    {** write network outputs into record output array **}
    n:=outstart;
    for i:=1 to outputsize do
      begin
        network.outdata^[i]:=n^.activation;
        n:=n^.next_node
      end;
    end;
  end;

procedure indata(network:nnet;index:integer;value:real);
{** load in an input data value **}
begin
  if (index>0) and (index<=network.inp) then

```

```

    network.indata^[index]:=value;
end;

function outdata(network:nnet;index:integer):real;
  {** retrieve a neural network output value **}
begin
  if (index>0) and (index<=network.out) then
    outdata:=network.outdata^[index];
  end;
end.

```

Other Neural Network Related Programs

Microsoft Excel was used for creating the artificial data sets, and for preparing the bank data sets. The results from the neural networks were also analysed using Excel.

A Pascal program *nndata* was used to scale the smaller data sets. It allows the data to be viewed at any point in the process, can scale from one range to another (creating a report file that stores the scaling information), randomise the data and move selected data to the end of the data set (this is used for validation data sets). It stores data in memory using ordinary arrays so it is limited in the amount of data that it can handle. The larger data sets were scaled using another Pascal program *lscale* which reads data directly from file, so it does not have storage limitations. This program also creates a scale report file.

A Pascal program *empprob* was used to create the empirical probabilities for the artificial continuous data sets described in Chapter 4.

Another Pascal program *testinvar* was used for testing the bank case study data sets involving the balk/stay decision that used the INVAR neural network approach as described in Chapter 6. For each combination of counters open and queue size, a series of probability values were created 0.005, 0.015, ..., 0.995, to determine the percentage of customers who would stay in the bank for each situation. The program used the simnet unit to run the neural network.

Appendix B

Bank Simulation Program

B1 : Program Structure

The Bank Simulation program was written in Borland Pascal 7 for DOS, with extensions using MicroSim library units. The program uses the 3-Phase method.

The program can be broken up into sections with general functionality. Following the order that they are found in the program, these are:

GENERAL PURPOSE ROUTINES

- additions to simulation units

SIMULATION SETUP

- definitions of entities and sets
- loading arrival and scenario data
- initial variable values
- graphical setup

SCENARIOS

- setting up a particular scenario
- running a data collection session

QUEUEING DECISIONS

- simple queueing decision - shortest queue

B-PHASE ROUTINES

- clock update
- customer arrivals
- services end
- next service decisions

C-PHASE ROUTINES

- start services

USER INTERFACE FOR DECISION MAKING

- control and graphics for highlighting counters

RECORDING DECISION DATA

special arrives

storing criteria and decisions made

SIMULATION LOGIC CONTROL

runs 3-phase method

B2 : Input Distributions

Arrival Distributions

Arrivals of customers (except user controlled customer) for each of the services follow a negative exponential distribution, with thinning for variable arrival rates at different times of the day.

The thinning procedure involves determining the maximum arrival rate for each of the services. When an arrival occurs, it is accepted with a probability of

$$\text{actual arrival rate} / \text{maximum arrival rate}$$

The arrival rates are stored in a file *arrivals.inp* and show the arrival rates per hour for each 15 minute segment of the day, as shown below:

Time	Arrival Rate /hour				Time	Arrival Rate /hour				Time	Arrival Rate /hour			
	Inf	Tra	Bus	Cur		Inf	Tra	Bus	Cur		Inf	Tra	Bus	Cur
9.30	20	50	10	10	12.00	10	50	10	10	14.30	15	45	15	10
9.45	15	60	10	5	12.15	15	60	15	15	14.45	15	45	10	10
10.00	10	50	10	5	12.30	15	70	15	15	15.00	10	45	5	5
10.15	10	50	10	5	12.45	15	80	15	15	15.15	10	45	10	5
10.30	10	50	10	5	13.00	20	80	15	15	15.30	10	45	15	10
10.45	10	60	15	5	13.15	20	90	10	15	15.45	10	45	15	10
11.00	20	50	10	10	13.30	15	70	10	15	16.00	15	50	20	10
11.15	10	50	10	10	13.45	15	60	10	15	16.15	5	30	15	5
11.30	10	50	05	10	14.00	20	50	10	5	16.30	0	0	0	0
11.45	10	50	10	10	14.15	20	50	15	10					

Service Distributions

Information Service : Gamma(2.29, 1.75) + 0.5

Transactions Service : Gamma(4, 0.5)

Business Service : Gamma (3, 0.75)

Information Service : Gamma(2.78, 0.9) + 0.5

After Service Distributions

After receiving their service some of the customer types could go onto other services rather than leaving the bank. These are the Information and Transactions customers. A customer will go to a maximum of one other service.

For Information customers, the probabilities are:

Leave : 0.5 Transactions : 0.25 Business : 0.15 Currency : 0.1

For Transaction customers, the probabilities are:

Leave : 0.98 Information : 0.02.

B3 : Entity Attributes

Customer entities have attributes which can be recorded for each one to help with logic control in the simulation. The attributes can only take integer values. Special customers (user controlled) use the same attributes as ordinary customer entities.

The attributes are :

Attribute No.	Representation	Values
1	Customer Type	1 = Information, 2 = Transactions, 3 = Business, 4 = Currency, 5 = Special
2	Service Required	1 = Information, 2 = Transactions, 3 = Business, 4 = Currency, 0 = leave
3	Queue Joined	1 - 5 - possible values depend on service

B4 : MicroSim Commands

The following is a guide to MicroSim routines that are available through the simkit, graphkit and simadd units. The notes were written by R.D. Hurion for teaching purposes at the University of Warwick.

VISUAL INTERACTIVE SIMULATION

<MICRO-VISION>

1. INTRODUCTION

These notes describe a visual interactive simulation system designed specifically for the micro-computer teaching environment at Warwick University.

'Visual Interactive Simulation' is the term applied to discrete event simulation which allows a user to watch the progress of a simulation model on a computer terminal. The user can then interact with the simulation model in order to try different experiments or strategies.

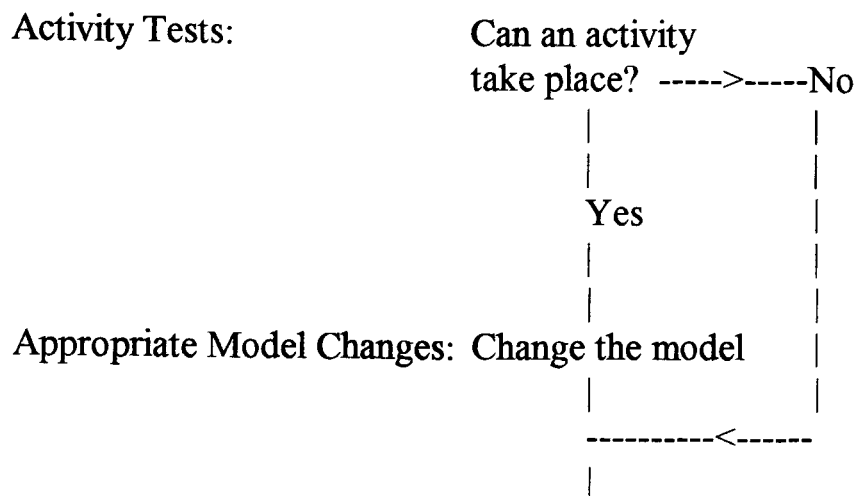
The facilities of <MICRO-VISION> exist as a set of high level PASCAL routines which are suitable for use with an IBM or IBM compatible micro-computer. The system has been implemented for the micro-computer work rooms at Warwick.

2. THE METHODOLOGY OF VISUAL INTERACTIVE SIMULATION

The methodology of the interactive simulation approach is to have:

- i) A simulation language in which it is possible to write complex industrial problems.
- ii) The ability of having a realistic one-to-one correspondence between elements in the model and their display on the screen.
- iii) Flexibility while running the model in order to change it and experiment with alternative configurations.

Event scheduling, process interaction and activity scanning are simulation methods that have been used for interactive model development. However the activity scan approach gives an easy method of model building and has distinct advantages for interaction at run time. The activity scan method consists of testing activities to see if they can occur. If an activity can occur then the appropriate change to the model is made. The typical structure for an activity is:



The activity tests contain the logic to ensure an activity can take place. If a one-to-one correspondence exists between elements in the model and elements on the visual display screen, then modification of activity tests is possible. This gives flexibility in the model structure at run time.

The micro computer screen should be considered as a set of windows through which the model can be 'observed'.

The best micro to use is one which has an VGA graphics screen. The actual computer screen has co-ordinates 0-639 in the horizontal (ix direction) and co-ordinates 0-479 vertically downwards (iy direction). An M24 Olivetti uses a 600X200 two colour grid. Elements may also be defined outside these grid co_ordinates but will not be shown on the screen.

If elements are defined within this grid then movement occurs automatically when the simulation model runs.

The grid described above should be considered as a 'window'. The system routines allow the user to develop models using up to 9 similar logical windows. Any one of these windows can be mapped to the micro-computer screen in order to show either parts of a complex model , a complete model or histograms.

The simulation facilities are implemented as high level PASCAL functions and procedures. The user will develop a PASCAL Simulation program which consists mainly of calls to system routines.

The routines for defining elements, their manipulation, (both within the model and on the screen), together with random and stochastic variate generation are facilities provided by the system. The system has the facility of an

own interaction facility, which allows for specific user defined interaction and displays.

3. SYSTEM STARTUP ROUTINE

setup;

This routine will initialise the simulation system. It will set the simulation time to zero, start all random number streams, set <logical display 1> on and <logical displays 2-9> off.

4. ELEMENT DEFINITION ROUTINES

Four different element types (entities, classes, sets and histograms) can be defined. The total number of individual elements should not total more than approx 800.

a)SET DEFINITION

The routine:-

```
define_set(set_name,sc,x,y,ix,iy);
```

will define an ordered set or queue where:-

- <set_name> is a set variable whose value is set by the routine. It MUST NOT be changed by the model. This variable acts as a pointer to the set.
- <sc> is an integer variable in the range 1-9. The value of <sc> denotes the logical screen upon which the members of the set will be displayed.
- <x> is an integer variable denoting the x co-ordinate of the set.
- <y> is an integer variable denoting the y co-ordinate of the set. (The y axis operates in a vertically downwards direction).
- <ix> are integer variables (usually set to -8,-4,0,4, ..).
- <iy> They give the x and y increment of new members joining the set. Each entity in the model will be shown using a grid of 8X8 pixels. The increment values show where the next element

in the set is to be drawn.

An example is:-

```
define_set(train_q,1,320,260,-2,8);
```

This instruction will define a queue which can be referenced in a model by using the variable <train_q>. Members of the queue will be displayed on screen 1 from position (320, 260). The i th member of the set will thus be shown at $(320-2i, 260+8i)$

b) ENTITY DEFINITION

The major type of element used in a discrete event model is the entity. It is used to model specific components.

```
define_entity(entity,des,forc,bacc);
```

will define a single entity in the simulation, where:-

<entity> is an entity variable whose value is set by the routine. It **MUST NOT** be changed by the model and is used as a pointer to the entity.

<des> is the entity character description.

<forc> is the foreground colour of <des>.

<bacc> is the background colour of <des>.

An example is:-

```
define_entity(loader,'l',red,green);
```

This instruction will define an entity referenced in the model by using the variable <loader>. The entity will be displayed as the character 'l' and will be shown using the colours red on green.

c) CLASS DEFINITION

Simulation models often have groups of similar entities. A customer entering a bank would be modelled as an entity, however, we may be interested in

the class of all customers entering the bank during (say) a busy lunch period,

```
define_class(customer,size,des,forc,bacc,sc,x,y,ix,iy);
```

will define a class of similar entities. The class is referenced by the class variable <customer>.

<size> is an integer variable defining the number of entities in the class (should be in the range 1-200).

<des,forc,bacc> have the same meaning as an entity and

<sc,x,y,ix,iy> have the same meaning as a set. Thus defining a class defines a group of similar entities and places them in a set with parameters <sc,x,y,ix,iy>. the set or class can then be referenced by <customer>.

An example is:-

```
define_class(merchant,200,'m',blue,cyan,1,10,-10,0,0);
```

This routine will define a class of merchants. Each merchant will be shown as 'm' using blue on cyan colours. All 'merchants' will be placed in a set at (10,-10) i.e. above the screen. The (x,y) increment is (0,0), so when merchants are used in the model they will scroll from their home (10,-10) position.

d)HISTOGRAM DEFINITION

The system allows the user to define histograms. Values may be added to histograms while the simulation progresses. The display, mean, and variance for each histogram is continuously updated by the system.

```
define_histogram(name,desc,sc,x,y,xsize,ysize,forc,bacc,
on_off,minx,maxx,cells);
```

will define a histogram, where:-

<name> is the histogram variable whose value is set by this procedure. This variable must not be changed by your model and acts as a pointer to the histogram.

<desc> is the text display name of the histogram

e.g. 'Coal Lorry Times'

- <x> are the top left co-ordinates of the
- <y> histogram.
- <xsize > is the screen horizontal size of the histogram.
- <ysize> is the screen vertical size of the histogram.
- <forc,bacc> are the foreground and background colours.
- <on_off> is a boolean variable which indicates if the
 histogram is being used to record values.
- <minx,maxx> are the minimum and maximum scale values of
 the histogram.
- <cells > is the number of cells in the histogram.
 (cells in the range 10 to 600).

An example this routine is:-

```
define_histogram(ncb_hist,'ncb_times',2,420,40,200,100,blue,cyan,
                true,0.0,500.0,100);
```

This routine will define a histogram referenced in the model by <ncb_hist>. The histogram will have the title 'ncb_times' and will be displayed with its top left vertex at co-ordinates (420,40) on screen number 2. The colour/display will be blue on cyan and recording will be set on. The histogram is defined for the range 0-500 with 100 cells. Each cell in this example will record values in units of 5.i.e.

```
0.000 - 4.9999 ... interval 1
5.000 - 9.999 ... interval 2
```

Two 'garbage' cells are also automatically defined with the histogram. They have ranges of:-

```
- inf < minx
and
maxx < + inf.
```

Any value when added to the histogram will increase one and only one cell count. An online total of the number of observations, mean and variance for each histogram is also maintained.

5. MODEL MANIPULATION ROUTINES

This section describes the routines which allow the status of a model to be changed.

a) `add_last(set_name,element);`

This routine will add a specific entity referenced by `<element>` to the end of the set referenced by `<set_name>`

b) `add_head(set_name,element);`

This routine will 'queue jump' and add the specific entity `<element>` at the head of the set `<set_name>`. Existing members of the set are displaced by one.

c) `delete(element,set_name);`

This routine will remove the element `<element>` from the set `<set_name>`. An error will occur if `<element>` is not a current member of `<set_name>`.

d) `set_attribute(entity,pos,value);`

Each entity has 5 integer attributes.
These attributes are allocated when the element is defined. This routine will set the attribute at position `<pos>` of the entity `<entity>` to the value `<value>`.

e) `move_on(element,set1,set2);`

All the previous set manipulation routines simply remove or add elements to sets. This routine `move_on` moves an element between sets but also allows scrolling (i.e. dynamic movement) on the screen.

The routine will take the element `<element>` from the set `<set1>` and move it in a straight line to the tail of the set `<set2>`.

[Note:- The routine first moves the element `<element>` to the head of the set `<set1>`. The co-ordinates (x1, y1) of this head element are obtained. The co-ordinates of tail of set `<set2>` (x2, y2) are also evaluated. The element `<element>` is then scrolled between these co-ordinates. This scrolling occurs if either or both the logical screens of `<set1>` or `<set2>` are on.

If both logical screens are off then no scrolling will occur. It is possible to have one of the sets with co-ordinate of (10, -5) i.e. off the screen. If an element moves from this set to one that is displayed on the screen then the element scrolls from the edge of the screen.

6. MODEL STATUS/INSPECTION ROUTINES

This section describes a series of FUNCTIONS which allow the model builder to inspect the current status of a model.

a) `n:=size_of(set_name);`

This integer function will return to <n> the current size of the set referenced by <set_name>.

b) `element:=head_of(set_name);`

This function will return to <element> the current head member of the set <set_name>. If the set is empty an error occurs.

c) `element:=tail_of(set_name);`

This function will return to <element> the current tail member of the set <set_name>. If the set is empty an error occurs.

d) `element:=identity(position,set_name);`

This function will return to <element> the element which is at position number <position> of <set_name>. If <position> is 1 then the head element is returned. If <position> is set to a large value (larger than the size of the set) then the tail element is returned. If the set is empty then an error occurs.

e) `n:=position(element,set_name);`

This integer function searches the set <set_name> for the member <element>. If the entity is in the set then the position of the entity will be returned as an integer (i.e. 1 = head element, 2 = next element etc...). If the entity <element>

is not in the set then a zero is returned.

f) ival:=attribute(element,pos);

This function returns the value of the integer attribute <pos> of <element>.

7. SIMULATION TIMING, SCHEDULING AND RANDOM VARIATE GENERATION

The simulation time advance and scheduling routines are based on real (not integer) values. The simulation may run from 0.0 up to time 2,147,483,647.0. It is suggested that models be developed using the three phase approach. For this, the following timing routines will be needed:-

a) advance(next_event_no,time,element);

This routine will advance the simulation to the next event. The integer <next_event_no> is returned with event number which is about to occur. The real variable <time> is the current simulation time returned, after the time advance has occurred. The pointer <element> is the entity returned associated with the current event number.

b) schedule(event,after_time,element);

This routine is used to schedule a future event. The event number <event> will be scheduled to occur at 'time + <after_time>'. The entity <element> is associated with this future event and is the pointer value returned by <element> from time advance when this event actually occurs.

c)RANDOM VARIATE GENERATION

The system can use up to 32 independent random number streams.

The stochastic variate routines available are:-

i) init_streams;

This routine re-sets all the random number streams to their initial values.

ii) `r:=rnd(s);`

This function will return a random number in the range 0.0-1.0 from stream <s>.

iii) `r:=negexp(m,s);`

This function will return a real sample from the negative exponential distribution with a mean of <m> from stream <s>.

(i.e. `r:=negexp(2.5,1);` will give the next random sample from the negative exponential distribution with mean 2.5 from stream 1).

iv) `r:=normal(m,sd,s);`

This function will return a sample from the normal distribution with real parameters mean <m> and standard deviation <sd> using stream <s>.

v) `r:=log_normal(m,sd,s);`

This function will return a sample from the log_normal distribution with parameters mean <m> std dev <sd> using stream <s>.

vi) `r:=uniform(a,b,s);`

This function returns a sample from the uniform distribution (a,b) using stream <s>.

vii) `i:=poisson(m,s);`

This function will return an integer sample from the poisson distribution with mean <m> using stream <s>.

viii) `i:=binomial(p,n,s);`

This routine will return an integer random sample from the binomial distribution using parameters <p,n> with stream <s>.

ix) `r:=gamma(m,k,s);`

This function returns a sample from the gamma distribution with mean <m> , shape <k> using stream <s>.

x) `r:=weibull(b,c,s);`

Returns a sample from the weibull with parameters `<b,c>` using stream `<s>`.

xi) `r:=chi_square(r,s);`

Returns a sample from the chi_square distribution with mean `<r>` using stream `<s>`.

xii) `r:=beta(a,b,s);`

Returns a sample from the beta distribution with parameters `<a,b>` using stream `<s>`.

xiii) `i:=geometric(p,s);`

Returns an integer sample from the geometric distribution with parameter `<p>` using stream `<s>`.

xiv) `i:=negative_binomial(k,p,s);`

Returns an integer sample from the negative binomial using parameters `<k,p>` from stream `<s>`.

Note. For the functions shown above that return a value to `r`, then a real valued sample is returned. If a value is returned to `i` then it is integer. The stream numbers can also be set in the range -1 to -32. These parameters give antithetic random variates.

8.SIMULATION DISPLAY ROUTINES

The majority of the simulation display routines are incorporated automatically in the simulation model manipulation facilities described earlier. However, it is important to remember that the output from a simulation can be directed to any one of 9 'logical windows'. In a similar manner user interaction with a model can also be via any of the logical windows. When the system is initialised, logical window 1 is set on, while the remaining windows are all set off.

The routines that can help with specific display handling are:-

a) `text_form(sc,x,y,text,dir,size,f_col,b_col);`

The routine will display the text held in <text> with foreground/background colours <f_col,b_col> at position <x>, <y>. This display will only occur if <sc> (the logical screen or window number) is currently on. If either x or y are outside the 'window' co-ordinates then the text will not be formed. The direction of the text <dir> can be horizontal <0> or vertical <1>. The size of the text <size> can be in the range 1-2. <1> gives the standard size. <2> gives large text.

An example of the routine is:-

```
text_form(5,6,23,'Supermarket',0,1,green ,black);
```

which will form the text 'Supermarket' at co-ordinates 6,23 using the colours green on red PROVIDING screen 5 is currently on. The text will be standard size.

b) `integer_form(sc,x,y,value,dir,size,f_col,b_col);`

This routine is similar to text_form. It will output the value of the integer <value> using the same criteria as text_form. The range of these integers are -32768 to 32767. If longer integers are required then :-

```
longint_form(sc,x,y,value,dir,size,f_col,b_col);
```

can be used. This routine will display integers in the range -2147483648 to 2147483647.

c) `real_form(sc,x,y,value,dir,size,f_col,b_col);`

This routine will display the real variable <value> using the same parameters as text_form.

d) `draw_line(sc,x1,y1,x2,y2,col);`

This routine will draw a line from (x1,y1) to (x2,y2) using the colour <col> providing logical screen number <sc> is on.

e) `draw_ellipse(sc,x,y,xrad,yrad,f_col,b_col);`

This routine is useful for filling in the static display or background for any model. The routine will fill in the ellipse at (x,y) using (xrad,yrad) in colours (f_col,b_col). The ellipse is drawn only if screen <sc> is on.

f) `draw_triangle(sc,x1,y1,x2,y2,x3,y3,f_col,b_col);`

This routine draws a solid triangle at (x1,y1) , (x2,y2), (x3,y3) using (f_col,b_col) providing screen <sc> is on.

g) `draw_bar(sc,x1,y1,x2,y2,col);`

This routine draws a solid rectangle between (x1,y1) and (x2,y2) using <col> providing screen <sc> is set on.

h) `change_description(element,ch,f_col,b_col);`

This routine is quite useful and will change the description of the entity <element> to the character <ch> using <f_col,b_col> as the colours. Note:- The pascal function chr(n) may be used in place of the character 'ch'. The full range of ascii characters may be drawn. i.e. chr(3) will give the 'hearts' symbol.

i) `cleardevice;`

This routine will clear the screen. (But NOT the current status or logic of the model).

j) `screen_on(sc);`

This routine will first turn all screens off. It will then turn screen number <sc> on.

k) `display_set(set_name);`

This routine will display the set <set_name>. This routine is used to help reform all the display screens after a user interaction.

l) `display_histogram(hist_name);`

This routine will display the histogram <hist_name>. It is also used to help reform the display after a user interaction.

9. RECORDING

A simulation is an experiment and so for any particular model configuration we have to decide what should be recorded, when and how. This section describes the recording facilities available in the visual interactive simulation system.

a) `record_in(hist,value);`

This routine will add the value of <value> to the histogram referenced by <hist>. The number of observations, mean and variance are updated. If the logical display of the histogram is on the the histogram will be updated.

b) `r:=rmean(hist);`

This function gives the current mean of the histogram <hist>.

c) `std:=stdev(hist);`

This function gives the current standard deviation of values in the histogram <hist>.

d) `n:=n_obs(hist);`

This function gives the number of observations in <hist>.

e) `timer_on(element);`

When comparing one simulation experiment with another, it is common to count or record how long elements/entities remain in the model. This routine `timer_on` sets a timer for the entity <element>.

e) `r:=timer_value(element);`

This routine gives the current value of the timer for <element>. The result can be added to a histogram. i.e.
`record_in(hist,timer_value(element);`

f) `timer_off(element);`

This routine will turn the individual time clock of `<element>` off and return its value back to zero.

e.g. consider the last 3 routines.

Suppose a customer enters a bank; then when this occurs `timer_on(cust)` will set the 'internal clock' running for the particular customer `<cust>`. When this customer leaves the bank the routine

`record_in(bank,timer_value(cust));` will record the time the customer `<cust>` spent in the bank where `<bank>` has been previously defined as a suitable histogram. Finally using `timer_off(cust)` will turn its own clock off.

The use of histograms for recording purposes is under the control of the simulation analyst. It is usually 'good simulation practice' to record the status of a model when it has reached a 'steady state' i.e. when the model is in a reasonably typical state. Three further routines are available. They are:-

g) `recording_on(hist);`

This routine will switch on the recording for `<hist>`.

h) `recording_off(hist);`

This routine will switch off the recording for `<hist>`.

i) `clear_histogram(hist);`

This routine will empty `<hist>` of all its values.

10. INTERACTIONS

The basic philosophy of visual interactive simulation is that it gives to a user the ability of watching the progress of a model. The user can then interact with the model in order to try different experiments or strategies. This section describes the interactions available:-

There are a standard set of interactions which can be used as the model is running. The interactions are obtained by using the function keys. They are:-

- `<F1>` continue running the model.
- `<F2>` own_interaction (see below).
- `<F3>` display histograms.

<F4>
 <F5> simulation trace on.
 <F6> simulation trace off.
 <F7> histogram display off.
 <F8>
 <F9> graphics off (batch run simulation).
 <F10>

The model can run at five different graphics speeds.
 The two keys <up_arrow> and <down_arrow> increase and decrease the speed of the model.

<END> key:-

Pressing the <END> key will stop the simulation.

Own_Interaction <F2> key

This interaction should be considered as the simulation analysts OWN interaction. It allows the analyst to set up 'bullet proof' interventions with the model WITHOUT running the risk of invalidating the model.

<F5> and <F6> Trace keys;

Pressing <F5> gives a trace of the simulation. It shows the number, entity and time of the next_event; The spare memory available is also shown.

<F3> and <F7> Histogram keys;

The system will update and display histograms if required. However it can slow down the execution of a model. Pressing <F7> turns histogram displays off, but they continue to record. <F3> turns histogram displays back on.

<F1> and <F9> Run keys;

<F1> will run the model in graphics mode. Pressing <F9> will turn the graphics off and run the model much faster in batch mode.

Input

The system allows the user to input text, integers and real values. The routines to do this are:-

```
input_text(x,y,'prompt',for_c,bac_c,text);
```

```
input_real(x,y,'prompt',for_c,bac_c,rval);
input_integer(x,y,'prompt',for_c,bac_c,ival);
```

where:-

<x,y> are the co_ordinates within the interaction screen for the string <'prompt'> using colours <for_c,bac_c>. The result of the input is returned to the variables text,rval,or ival.

An example is :-

```
input_real(10,95,'Merchant Rate ',yellow,black,arm);
```

This procedure will cause the prompt 'Merchant Rate' to appear on the interaction screen using colours yellow on black. The procedure will wait until a valid response (real number) has been typed. The variable <arm> will contain this value.

Colours

The standard Turbo Pascal colours are used. The colour codes are shown below together with their numeric value (shown in brackets). If you are using a mono_screen computer then use odd and even colour combinations to obtain a contrast.

Black (0)	Blue (1)	Green (2)	Cyan (3)
Red (4)	Magenta (5)	Brown (6)	LightGray (7)
Darkgray(8)	LightBlue(9)	LightGreen(10)	LightCyan(11)
LightRed(12)	LightMagenta(13)	Yellow (14)	White(15)

Summary of Simkit Routines

```

procedure change_description(element:entity;ch:char;forc,bacc:integer);
function description(element:entity) : char;
function foreground_colour(element:entity) : colour;
function background_colour(element:entity) : colour;
function simulating :boolean;
function head_of(s:sets) :entity;
function tail_of(s:sets) :entity;
function identity(pos:integer; s:sets) : entity;
function position(element:entity; s:sets) : integer;
function create_entity(des:char;forc,bacc:colour) : entity;
procedure destroy_entity(element:entity);
function size_of(s:sets) :integer;
function timer_value(element : entity) : real;
procedure timer_on(element : entity);
procedure timer_off(element : entity);
procedure define_entity(var name:entity;des:char;forc,bacc:colour);
procedure set_attribute(name:entity;pos:integer;value:integer);
function attribute(name:entity; pos:integer) : integer;
procedure define_class(var c:class;size:integer;des:char;forc,bacc:colour;
                        sc,x,y,ix,iy:integer);
procedure define_set(var set_name:sets; sc,x,y,ix,iy: integer);
procedure add_head(set_name:sets; element:entity);
procedure add_last(set_name:sets; element:entity);
procedure delete(element:entity;set1:sets);
procedure schedule(event:integer; after_time:real; element:entity);
procedure display_set(s:sets);
procedure screen_on(k:integer);
procedure display_time(time:real;x,y:integer);
procedure move_on(element:entity; set1,set2:sets);
procedure advance(var next_event_no:integer; var time:real;
                  var element:entity);

procedure setup;
procedure open_own;
procedure clear_own;
procedure close_own;

```

Summary of Graphkit Routines

```

procedure text_form(sc,x,y:integer; s:string; dir,size:integer;
                   f_col,b_col:colour);
procedure integer_form(sc,x,y:integer; value,dir,size:integer;
                      f_col,b_col:colour);
procedure longint_form(sc,x,y:integer; value : longint; dir,size:integer;
                      f_col,b_col:colour);
procedure real_form(sc,x,y:integer; value:real; dir,size:integer;
                   f_col,b_col:colour);
procedure longreal_form(sc,x,y:integer; value:real; dir,size:integer;

```

```

        f_col,b_col:colour);
procedure draw_line(sc,x1,y1,x2,y2:integer; col:colour);
procedure draw_ellipse(sc,x,y,xrad,yrad:integer; for_c,bac_c:colour);
procedure draw_triangle(sc,x1,y1,x2,y2,x3,y3:integer; for_c,bac_c:colour);
procedure input_text(x,y:integer; prompt:string;
        f_col,b_col:colour; var sg:string);
procedure input_integer(x,y:integer; prompt:string;
        f_col,b_col:colour; var ival:integer);
procedure input_real(x,y:integer; prompt:string;
        f_col,b_col:colour; var rval:real);
procedure define_histogram(var name:histogram;desc:titlesize;
        sc,x,y,xsize,ysize:integer; col_f,col_b:colour;
        on_off:boolean; minx,maxx:real;
        cells:integer);
procedure draw_bar(sc,xstart,ystart,xfin,yfin:integer;col:colour);
procedure display_histogram(h:histogram);
procedure record_in(h:histogram; value:real);
procedure recording_on(h:histogram);
procedure recording_off(h:histogram);
procedure clear_histogram(h:histogram);
function rmean(h:histogram) : real;
function n_obs(h:histogram) : integer;
function stdev(h:histogram) : real;
procedure init_streams;
function rnd(s:integer) : real;
function uniform(a,b:real;s:integer) : real;
function normal(m,sd : real; s:integer) :real;
function log_normal(m,sd:real; s:integer) :real;
function poisson(m:real; s:integer) : integer;
function negexp(m:real;s:integer) :real;
function gamma(a,b:real;s:integer):real;
function triangular(a,b,c:real;s:integer):real;
{** a,b are min & max, c is most common (mode) **}
function beta(min,max,a,b:real;s:integer):real;
function weibull(b,c:real ; s:integer) : real;
function chi_square(r,s:integer) : real;
function geometric(p:real; s:integer) : integer;
function negative_binomial(k:integer; p:real; s:integer) : integer;
function binomial(p:real; n,s:integer) : integer;

```

Summary of Simadd Routines

```

procedure beep;
procedure shadow_box(scr,x1,y1,x2,y2:integer;col:colour);
procedure raised_box(scr,x1,y1,x2,y2:integer;col:colour);
function edit_string(sc,x,y,maxx:integer;str:string;forc,bak:colour):string;
function edit_real(sc,x,y,maxx:integer;r:real;forc,bak:colour):real;
function edit_integer(sc,x,y,maxx:integer;ival:integer;forc,bak:colour):integer;
function getfilename(sc,x,y,maxx:integer;defstr:string):string;

```


B5 : Bank Simulation Program Listing

```

{$R+}
program bank;

uses simkit,graphkit,simadd,crt,graph,dos;

type
  status = (open,closed,transfer,allocated,error);  { * counter status * }
  arrival_data_array = array[1..8] of integer;      { * recording arrival decisions * }
  q_data_array = array[1..800,1..11] of real;       { * recording queuing decisions * }

var
  info:array[1..4]of status;      { * status of information counters * }
  currency:array[1..2] of status; { * status of currency counters * }
  business:array[1..2] of status; { * status of business counters * }
  transactions:array[1..5] of status; { * status of transaction counters * }

  hours,mins:integer;            { * clock variables * }

  { **classes** }
  trans_cust:class;             { * transactions customers * }
  info_cust:class;              { * information customers * }
  curr_cust:class;              { * currency customers * }
  bus_cust:class;               { * business customers * }
  special:class;                { * human controlled customers * }
  staff:class;                  { * staff entities * }
  current_special:entity; { * label to keep track of special customer * }
  clock_dummy:entity;           { * dummy entity for update clock events * }
  period:entity;                { * dummy to update thinning periods * }
  branch:entity;                { * dummy to schedule opening/closing branch * }

  { ** sets - logical and graphical locations ** }
  outdoor,indoor,decide:sets;   { * customer dummy sets * }
  counter_out,counter_in,corner:sets; { * staff dummy sets * }
  info_q:sets;                  { * queue for information * }
  info_c:array[1..4] of sets;    { * information counters - customers * }
  info_s:array[1..4] of sets;    { * information counters - staff * }
  curr_q:array[1..2] of sets;    { * currency queues * }
  curr_c:array[1..2] of sets;    { * currency counters - customers * }
  curr_s:array[1..2] of sets;    { * currency counters - staff * }
  bus_q:array[1..2] of sets;     { * business queues * }
  bus_c:array[1..2] of sets;     { * business counters - customers * }
  bus_s:array[1..2] of sets;     { * business counters - staff * }
  trans_q:array[1..5] of sets;   { * transaction queues * }
  trans_c:array[1..5] of sets;   { * transaction counters - customers * }
  trans_s:array[1..5] of sets;   { * transaction counters - staff * }
  to_trans,to_currency,to_business:sets; { * customer dummy sets * }

  { * counter staff variables * }
  { * indicate which counters are to be closed, so no more customers can * }
  { * join queue - disabled in bank game * }
  counters_clear:array[1..13,1..2] of integer;
  num_counters_clear:integer;

  { * arrival rates * }

```

```

arr_rate:array[1..4,1..29] of integer;  { * arrival rates for thinning * }
max_rate:array[1..4] of integer;    { * maximum arrival rate - I,T,B,C * }
arr:array[1..4] of integer;         { * current arrival rates * }
arrivals:text;                      { * file to load in arrival rates * }

{ * branch opening logic * }
branch_open:boolean;  { * specifies if branch is open/closed * }
shut:boolean;        { * specifies when branch shut & all customers gone * }
num_cust:integer;    { * keeps track of number of customers in branch * }

{ * game scenarios * }
scenef:text;         { * file to load in scenario data * }
scenestr:string;     { * scenario data filename * }
spec_arr:real;       { * simulation time for arrival of special * }
setup_stage:boolean; { * prevents customer arrivals until scenario setup * }
scenario:boolean;    { * specifies that scenario is running * }
special_in:boolean;  { * specifies when special customer is in the system * }
clock_delay:integer; { * used to slow down model during scenario * }
show_no:integer;     { * keeps track of scenario number * }

{ * scenario & game details * }
spec_type:integer;   { * service required by special customer * }
total_time,current_time:real; { * total & current queue time scores * }
balk_penalty:real;   { * exit penalty for current scenario * }
balk:boolean;        { * specifies when special customer has balked * }
timing:boolean;       { * specifies that special customer is queuing * }
balk_store:array[1..5] of real; { * set of possible exit penalties * }

{ ** data collection variables ** }
arrive_data:arrival_data_array;
{ ** [1]=func, [2]=counters open, [3-7]=q per counter (-1=closed) ** }
{ ** [8]=q chosen (0=balk) ** }
q_data:q_data_array;
{ ** [1]=func, [2]=counters open, [3-7]=q per counter ** }
{ ** [8]=current q, [9] position in q, [10]=time in q, [11]=q chosen ** }

amt_q_dat:integer; { * number of queuing data items collected * }
subject_no:string; { * number of subject - used in data filenames * }
arr_res_f_name:string; { * arrival decisions data filename * }
q_res_f_name:string; { * in-queue decisions data filename * }
arr_df:text;         { * arrival data file * }
q_df:text;           { * in queue data file * }

procedure get_q_data;forward:
{ ** gets data about current queueing position of special customer - after ** }
{ ** checking that special customer is in queue, calls record_q_data ** }
{ ** used in C - phase, can be found in section of program concerning ** }
{ ** special customers. ** }

procedure record_q_data(element:entity);forward:
{ ** gets data about current queueing position of special customer ** }
{ ** if called directly it does not check that customer is in queue ** }
{ ** called directly to record when special customer is served ** }
{ ** used in C - phase, can be found in section of program concerning ** }
{ ** special customers. ** }
{ **** }

```



```
{** GENERAL ROUTINES **}
```

```
procedure hidden(element:entity;set1,set2:sets);
{** moves an entity from one set to another without showing transfer route **}
{** on the screen *}
begin
  delete(element,set1);
  add_last(set2,element)
end;
```

```
{*****}
```

```
procedure fill_in(sc:integer;x,y:integer;rimcol,fillcol:colour);
{** a friendly floodfill routine, leaves background colour the same **}
{** as before (unlike the built-in Pascal routine **}
var
  fillsettings:fillsettingstype;
```

```
begin
  if screens[sc] then    {*** if on the correct screen then ***}
  begin
    getfillsettings(fillsettings);  {*** remember current settings ***}
    setfillstyle(solidfill,fillcol); {*** set required settings ***}
    floodfill(x,y,rimcol);
    with fillsettings do
      setfillstyle(pattern,color);  {*** restore fill settings ***}
    end;
  end;
```

```
{*****}
```

```
function pos_normal(m,std:real;s:integer):real;
{** generates a duration from a normal distribution, throwing away **}
{** any negative values **}
var
  res:real;
begin
  repeat
    res:=normal(m,std,s)
  until res>0;
  pos_normal:=res
end;
```

```
{*****}
```

```
function thinning(cust:integer):boolean;
{** determines whether to accept or reject a thinned arrival distribution **}
begin
  if rnd(15)<(arr[cust]/max_rate[cust]) then
    thinning:=true
  else
    thinning:=false
  end;
```

```
{*****}
```

```

{** SIMULATION SET-UP PROCEDURES **}

procedure my_init_streams(add:integer);
{** initialises random number streams, with an increment value, so **}
{** that streams can be initialised to different starting points **}
{** this is used at the start of each scenario, to make sure that the **}
{** random numbers are consistent for each scenario and are not **}
{** side-effected by previous scenarios. The purpose of ADD is to **}
{** make sure that there is variety between scenarios, but is consistent **}
{** for separate runs of the program. ADD is the (truncated) arrival time *}
{** of the special customer **}
var
  i:integer;
begin
  for i:=1 to 32 do
    seed[i]:=i*1000+add; {** seed is a graphkit variable **}
  end;

  {*****}

procedure define_entities;
{** specify class, entity and set details **}
var
  i:integer;

begin
  {** dummy entities - used for scheduling simulation control events **}
  define_entity(clock_dummy,'d',black,black);
  define_entity(period,'d',black,black);
  define_entity(branch,'d',black,black);
  {** classes - collections of entities and specifies storage set**}
  define_class(trans_cust,100,chr(1),white,black,1,-10,206,0,0);
  define_class(info_cust,100,chr(1),white,black,1,-10,216,0,0);
  define_class(curr_cust,50,chr(1),white,black,1,-10,226,0,0);
  define_class(bus_cust,50,chr(1),white,black,1,-10,236,0,0);
  define_class(special,5,'*',lightred,white,1,-10,246,0,0);
  define_class(staff,14,'S',white,blue,1,650,206,0,0);
  {** sets - logical and graphical locations for entities **}
  define_set(outdoor,1,46,206,-8,0);
  define_set(indoor,1,86,206,8,0);
  define_set(decide,1,296,206,-8,0);
  define_set(counter_out,1,496,116,-8,0);
  define_set(counter_in,1,546,96,8,0);
  define_set(corner,1,546,346,8,0);
  define_set(info_q,1,314,172,-8,0);
  for i:=1 to 4 do
    begin
      define_set(info_c[i],1,233+(i-1)*50,142,0,8);
      define_set(info_s[i],1,225+(i-1)*50,116,8,0)
    end;
  for i:=1 to 2 do
    begin
      define_set(curr_q[i],1,210+(i-1)*40,296,-8,-8);
      define_set(curr_c[i],1,202+(i-1)*40,312,0,-8);
      define_set(curr_s[i],1,194+(i-1)*40,343,8,0)
    end;
  for i:=1 to 2 do
    begin

```

```

    define_set(bus_q[i],1,400+(i-1)*40,296,-8,-8);
    define_set(bus_c[i],1,392+(i-1)*40,312,0,-8);
    define_set(bus_s[i],1,384+(i-1)*40,343,8,0)
end;
for i:=1 to 5 do
begin
    define_set(trans_q[i],1,496,128+(i-1)*42,-8,4-(i-1)*2);
    define_set(trans_c[i],1,512,132+(i-1)*40,-8,0);
    define_set(trans_s[i],1,542,124+(i-1)*40,0,8)
end;
define_set(to_trans,1,368,206,-8,0);
define_set(to_currency,1,220,228,8,-8);
define_set(to_business,1,342,228,-8,-8);

end;

{ **** }

procedure load_arrival_rates;
{** load customer arrival rates from file **}
var
    i,j:integer;
begin
    max_rate[1]:=0; max_rate[2]:=0;
    max_rate[3]:=0; max_rate[4]:=0;

    assign(arrivals,'arrivals.inp');
    reset(arrivals);
    for i:=1 to 29 do
        begin
            for j:=1 to 4 do
                begin
                    read(arrivals,arr_rate[j,i]);
                    if arr_rate[j,i] > max_rate[j] then
                        max_rate[j]:=arr_rate[j,i]
                    end;
                end;
            readln(arrivals)
        end;
    close(arrivals)
end;

{ **** }

procedure clear_staff;
{** removes staff from allocated positions in bank **}
var
    i:integer;
begin
    {** move staff from counter sets to store of staff entities **}

    for i:=1 to 4 do
        if info[i] = open then
            begin
                element:=head_of(info_s[i]);
                hidden(element,info_s[i],staff);
            end;

```



```

for i:=1 to 2 do
  if currency[i] = open then
    begin
      element:=head_of(curr_s[i]);
      hidden(element,curr_s[i],staff);
    end;
for i:=1 to 2 do
  if business[i] = open then
    begin
      element:=head_of(bus_s[i]);
      hidden(element,bus_s[i],staff);
    end;
for i:=1 to 5 do
  if transactions[i] = open then
    begin
      element:=head_of(trans_s[i]);
      hidden(element,trans_s[i],staff);
    end;

```

```

{** specify all counters as closed **}

```

```

for i:=1 to 4 do
  info[i]:=closed;
for i:=1 to 2 do
  currency[i]:=closed;
for i:=1 to 2 do
  business[i]:=closed;
for i:=1 to 5 do
  transactions[i]:=closed;
end;

```

```

{ **** }

```

```

procedure read_servers;

```

```

{** read server allocations from scenario file **}

```

```

var

```

```

  c_state:array[1..13] of integer;

```

```

  i:integer;

```

```

begin

```

```

  for i:=1 to 13 do

```

```

    read(scenef,c_state[i]);    {** 1 = open , 0 = closed **}

```

```

  for i:=1 to 4 do

```

```

    if c_state[i]=1 then

```

```

      info[i]:=open

```

```

    else

```

```

      info[i]:=closed;

```

```

  for i:=5 to 9 do

```

```

    if c_state[i]=1 then

```

```

      transactions[i-4]:=open

```

```

    else

```

```

      transactions[i-4]:=closed;

```

```

  for i:=10 to 11 do

```

```

    if c_state[i]=1 then

```

```

      business[i-9]:=open

```

```

    else

```

```

      business[i-9]:=closed;

```

```

  for i:=12 to 13 do

```



```

    end;
end;

{*****}

procedure initial_conditions;
{** initialise branch state **}
var
  i:integer;
  element:entity;
begin

  num_counters_clear:=0;    {** all counters closed **}
  for i:=1 to 13 do
    begin
      counters_clear[i,1]:=0;
      counters_clear[i,2]:=0
    end;

  num_cust:=0; {** no customers in bank **}

  {** initialise exit penalty values **}

  balk_store[1]:=2;
  balk_store[2]:=5;
  balk_store[3]:=8;
  balk_store[4]:=11;
  balk_store[5]:=14;

  special_in:=false; {** no special cust in system **}
end;

{*****}
{** GRAPHICAL DISPLAY AND UPDATE ROUTINES **}

procedure display_clock_time;
{** displays 24 hour clock on top-right of screen **}
var
  hour_text,min_text:string;
begin
  str(hours:0,hour_text);
  if hours<10 then
    hour_text:='0'+hour_text;
  str(mins:0,min_text);
  if mins<10 then
    min_text:='0'+min_text;
  text_form(1,500,12,hour_text,0,2,white,black);
  text_form(1,548,12,min_text,0,2,white,black);
end;

procedure display_scores;
{** displays players score on top left of screen **}
var
  s:string;
begin
  str(total_time:5:1,s);
  text_form(1,192,12,s,0,2,white,black);

```

```

str(current_time:4:1,s);
text_form(1,208,42,s,0,2,white,black);
end;

```

```

procedure display_information(p:integer);
{** displays specified information desk as open or closed **}
var
  col:colour;
begin
  if info[p]=open then
    col:=green
  else if info[p]=closed then
    col:=lightred
  else
    col:=yellow;
  draw_bar(1,220+(p-1)*50,126,250+(p-1)*50,140,col)
end;

```

```

procedure display_transaction(p:integer);
{** displays transaction counter barrier as being open or closed for **}
{** the specified position **}
var
  col:colour;
begin
  if transactions[p]=open then
    col:=white
  else if transactions[p]=closed then
    col:=lightred
  else
    col:=yellow;
  draw_bar(1,529,120+40*(p-1),531,150+40*(p-1),col)
end;

```

```

procedure display_business(p:integer);
{** displays business counter barrier as being open or closed for **}
{** the specified position **}
var
  col:colour;
begin
  if business[p]=open then
    col:=white
  else if business[p]=closed then
    col:=lightred
  else
    col:=yellow;
  draw_bar(1,380+40*(p-1),329,410+40*(p-1),331,col)
end;

```

```

procedure display_currency(p:integer);
{** displays currency counter barrier as being open or closed for **}
{** the specified position **}
var
  col:colour;
begin
  if currency[p]=open then

```

```

    col:=white
else if currency[p]=closed then
    col:=lightred
else
    col:=yellow;
draw_bar(1,190+40*(p-1),329,220+40*(p-1),331,col)
end;

```

```

procedure form_screens;
{** set background graphics **}
var
    i:integer;
    col:colour;
begin
    cleardevice;

    integer_form(1,540,464,show_no,0,1,cyan,black);

    {** outline **}
    draw_bar(1,58,78,582,80,lightgray);
    draw_bar(1,580,78,582,382,lightgray);
    draw_bar(1,582,380,138,382,lightgray);
    draw_bar(1,138,382,140,330,lightgray);
    draw_bar(1,140,330,58,332,lightgray);
    draw_bar(1,58,332,60,220,lightgray);
    draw_bar(1,58,200,60,78,lightgray);
    {** clock box **}
    raised_box(1,494,7,584,30,black);
    text_form(1,532,12,':',0,2,white,black);
    display_clock_time;
    {** information sign **}
    raised_box(1,217,87,401,113,blue);
    text_form(1,221,94,'INFORMATION',0,2,white,blue);
    {** currency sign **}
    raised_box(1,156,353,295,379,blue);
    text_form(1,162,360,'CURRENCY',0,2,white,blue);
    {** business sign **}
    raised_box(1,347,353,484,379,blue);
    text_form(1,354,360,'BUSINESS',0,2,white,blue);
    {** cash tills sign **}
    raised_box(1,552,120,579,322,blue);
    text_form(1,559,127,'T',0,2,white,blue);
    text_form(1,559,143,'R',0,2,white,blue);
    text_form(1,559,159,'A',0,2,white,blue);
    text_form(1,559,174,'N',0,2,white,blue);
    text_form(1,559,191,'S',0,2,white,blue);
    text_form(1,559,207,'A',0,2,white,blue);
    text_form(1,559,223,'C',0,2,white,blue);
    text_form(1,559,239,'T',0,2,white,blue);
    text_form(1,559,255,'T',0,2,white,blue);
    text_form(1,559,271,'O',0,2,white,blue);
    text_form(1,559,287,'N',0,2,white,blue);
    text_form(1,559,303,'S',0,2,white,blue);
    {** partition walls **}
    draw_line(1,140,330,530,330,lightgray);
    draw_line(1,530,330,530,110,lightgray);
    draw_line(1,530,90,530,80,lightgray);

```



```

    {** information desks **}
for i:=1 to 4 do
    display_information(i);
    {** currency counters **}
for i:=1 to 2 do
    begin
        draw_bar(1,190+40*(i-1),320,220+40*(i-1),340,white);
        display_currency(i);
    end;
{** business counters **}
for i:=1 to 2 do
    begin
        draw_bar(1,380+40*(i-1),320,410+40*(i-1),340,white);
        display_business(i);
    end;
{** transaction counters **}
for i:=1 to 5 do
    begin
        draw_bar(1,520,120+40*(i-1),540,150+40*(i-1),white);
        display_transaction(i);
    end;

{** main door **}
draw_bar(1,58,200,59,220,white);
{** counter door **}
draw_line(1,530,90,530,110,brown);

{** message panel **}
raised_box(1,100,430,540,470,lightgray);
text_form(1,288,432,'MESSAGES',0,1,darkgray,lightgray);

{** score panel **}
raised_box(1,0,7,280,30,black);
text_form(1,5,12,'TOTAL TIME',0,2,white,black);
raised_box(1,0,37,280,60,black);
text_form(1,5,42,'CURRENT TIME',0,2,white,black);
display_scores;

{** display sets **}
display_set(outdoor);
display_set(indoor);
display_set(counter_out);
display_set(counter_in);
display_set(decide);
display_set(corner);
display_set(info_q);
for i:=1 to 4 do
    begin
        display_set(info_c[i]);
        display_set(info_s[i])
    end;
for i:=1 to 2 do
    begin
        display_set(curr_q[i]);
        display_set(curr_c[i]);
        display_set(curr_s[i])
    end;
for i:=1 to 2 do

```

```

begin
  display_set(bus_q[i]);
  display_set(bus_c[i]);
  display_set(bus_s[i])
end;
for i:=1 to 5 do
  begin
    display_set(trans_q[i]);
    display_set(trans_c[i]);
    display_set(trans_s[i])
  end;
end;

{*****}

```

```

procedure open_door;
{** blanks out closed bank door, and draws an open one **}
begin
  draw_bar(1,58,200,59,220,black);
  draw_bar(1,58,200,78,201,white)
end;

```

```

procedure close_door;
{** blanks out an open bank door, and draws a closed one **}
begin
  draw_bar(1,58,200,78,201,black);
  draw_bar(1,58,200,59,220,white)
end;

```

```

procedure open_counter;
{** blanks out a closed counter door, and draws an open one **}
begin
  draw_line(1,530,90,530,110,black);
  draw_line(1,530,90,510,90,brown)
end;

```

```

procedure close_counter;
{** blanks out an open counter door, and draws a closed one **}
begin
  draw_line(1,530,90,510,90,black);
  draw_line(1,530,90,530,110,brown)
end;

```

```

procedure write_message(s1,s2,s3:string,col:colour);
{** writes instruction message on bottom of screen **}
var
  pos1,pos2,pos3:integer;
begin
  raised_box(1,100,430,540,470,lightgray);
  pos1:=320-(length(s1)*8) div 2;
  pos2:=320-(length(s2)*8) div 2;
  pos3:=320-(length(s3)*8) div 2;
  text_form(1,pos1,432,s1,0,1,col,lightgray);
  text_form(1,pos2,445,s2,0,1,col,lightgray);
  text_form(1,pos3,458,s3,0,1,col,lightgray);
end;

```

```

procedure clear_message;

```

```

{** clears instruction message **}
begin
  raised_box(1,100,430,540,470,lightgray);
  text_form(1,288,432,'MESSAGES',0,1,darkgray,lightgray);
end;
{*****}
{** SCENARIO SPECIFICATION PROCEDURES **}

procedure set_scenario;
{** sets a scenario to run **}
var
  period_no:integer;
  no_info,no_trans,no_bus,no_curr:integer;
  i:integer;
  balk_no:integer;
  sub_no,c:integer;
  add:integer;
begin
  scenario:=true;
  initial_conditions;
  setup_stage:=true; {** normal arrivals cannot occur **}
  branch_open:=true;
  shut:=false;
  time:=spec_arr;
  hours:=9+((trunc(spec_arr+30)) div 60);
  mins:=trunc(time+30) mod 60;
  amt_q_dat:=0; {** no data collected on queuing decisions yet **}
  current_time:=0;
  balk:=false;
  timing:=false;
  schedule(99,1,clock_dummy);
  clock_delay:=0;
  inc(show_no);

  period_no:=1 + trunc(time) div 15;
  set_attribute(period,1,period_no);

  {** set the initial customer numbers for each function **}
  {** schedule arrivals so customers in at start of sim **}
  {** adjust initial arrival rates so all customers **}
  {** accepted during thinning **}
  read(scenef,spec_type,spec_arr,no_info,no_trans,no_bus,no_curr);

  {** initialise random number streams with ADD of time of special **}
  {** arrival - this ensures that each scenario has different random **}
  {** number, but they are consistent between different runs of model **}
  add:=trunc(spec_arr);
  my_init_streams(add);

  for i:=1 to 4 do
    arr[i]:=max_rate[i];

  for i:=1 to no_trans do
    schedule(1,0,identity(i,trans_cust));
  for i:= 1 to no_info do
    schedule(2,0,identity(i,info_cust));
  for i:=1 to no_curr do
    schedule(3,0,identity(i,curr_cust));

```



```

for i:=1 to no_bus do
  schedule(4,0,identity(i,bus_cust));

{** read server information from same file as customers **}
read_servers;
readln(scenef); {** new line for next scenario **}
form_screens;
display_on:=true;
set_servers;
display_on:=false;

if spec_arr>0 then
  begin
    schedule(203,0.1,branch);
    {** set balk penalty for this case **}
    {** penalty number from hash number based on the special arrival time **}
    {** and the subject number. Subject number divided since it is odd **}
    {** for all scene1 and even for scene2, so it needs correcting to **}
    {** give odd and even values for both scenes so as to give the full **}
    {** of balk values : 0=1, 1=1, 2=2, 3=2, 4=3, 5=3, 6=4, 7=4, 8=5, 9=5 **}
    val(subject_no,sub_no,c);
    balk_no:= 5- (trunc(spec_arr) + (sub_no div 2 +1)) mod 5;
    balk_penalty := balk_store[balk_no];
    {** [1]=2, [2]=5, [3]=8, [4]=11, [5]=14 **}
  end
else
  scenario:=false;
end;

```

```

procedure end_scenario;
{** end of scenario, write results to file, and schedule next scenario **}
var
  i,j:integer;
begin
  clear_staff;
  write(arr_df,balk_penalty:4:0);
  for i:= 1 to 8 do
    write(arr_df,arrive_data[i]:4);
  writeln(arr_df);
  for j:=1 to amt_q_dat do
    begin
      write(q_df,balk_penalty:4:0);
      for i:=1 to 11 do
        write(q_df,q_data[j,i]:6:2);
      writeln(q_df);
    end;
  total_time:=total_time+current_time;
  if balk then
    total_time:=total_time+balk_penalty;
  schedule(201,0,branch);
end;

```

```

procedure start_scenario;
{** schedules arrivals next arrivals. Note that special - the customer **}
{** controlled by the user - is the next arrival **}
begin
  setup_stage:=false;

```



```

schedule(100,0.1,period);
schedule(1,1,head_of(trans_cust));
schedule(2,1.3,head_of(info_cust));
schedule(3,1.5,head_of(curr_cust));
schedule(4,1.6,head_of(bus_cust));

```

```

schedule(30,0.5,head_of(special));
end;

```

procedure session;

```

{** opening screen to set subject code, scenario number and balk penalty **}
{** starts scenarios **}

```

var

c:char;

begin

total_time:=0;

current_time:=0;

show_no:=0;

open_own;

scenestr:='practice';

text_form(1,20,40,'Change scenario file using <SPC>, Select with <RTN>',0,1,red,cyan);

text_form(1,80,60,scenestr,0,1,red,white);

repeat

c:=readkey;

if c=' ' then

begin

if scenestr='practice' then

scenestr:='scenel'

else if scenestr='scenel' then

scenestr:='scene2'

else

scenestr:='practice'

end;

text_form(1,80,60,' ',0,1,white,white);

text_form(1,80,60,scenestr,0,1,red,white)

until c=chr(13); {** return key **}

text_form(1,20,80,'Enter Subject No :',0,1,red,cyan);

subject_no:=edit_string(1,188,80,228,"",red,white);

close_own;

arr_res_f_name:='ar_res'+subject_no+'.out';

q_res_f_name:='q_res'+subject_no+'.out';

assign(scenef,scenestr+'.inp'); {** customer/server nos **}

reset(scenef);

assign(arr_df,arr_res_f_name); {** arrival decisions **}

assign(q_df,q_res_f_name); {** queueing decisions **}

rewrite(arr_df);

rewrite(q_df);

set_scenario;

end;

```

{*****}

```

```

{** OWN INTERACTION **}

```

procedure own_interaction;

```

{** allows simulation parameters to be altered during run time **}

```

```

{** note. for the simulation game, no own_interaction is allowed **}

```

```

{** However, the routine is called by a function built into SIMKIT **}
{** so a routine (that does nothing in this case) must be specified **}
var
  i,option,screen_no:integer;
  ch:char;
  finish:boolean;
begin
  finish:=false;
  open_own;
  screen_no:=1;
  screen_on(screen_no);
  repeat
    {** temporary **} finish:=true;
  until finish=true;
  screen_on(screen_no);
  close_own;
  form_screens;
end;

{*****}
{** SERVICE_DECISION ROUTINES **}

{** simple queue joining routines - random choice of shortest queues **}

procedure shortest_transactions(element:entity);
{** see which transaction_q to join - at least 1 counter **}
{** must be open **}
var
  i,q,best,shortest,r_int:integer;
begin
  {** randomize which q to look at first **}
  best:=0; {** stops range error if none trans open **}
  shortest:=9999;
  r_int:=trunc(5*rnd(5));
  for i:=1 to 5 do
    begin
      q:=1+(r_int+i) mod 5;
      if (transactions[q]=open)and
        ((size_of(trans_q[q])+size_of(trans_c[q]))<shortest) then
        begin
          best:=q;
          shortest:=size_of(trans_q[q])+size_of(trans_c[q])
        end
      end;
    set_attribute(element,3,best);
  end;

procedure shortest_currency(element:entity);
{** see which currency_q to join - at least 1 counter **}
{** must be open **}
var
  i,r_int,best,shortest,q:integer;
begin
  best:=0;
  shortest:=9999;
  r_int:=trunc(2*rnd(12));
  for i:=1 to 2 do
    begin

```

```

q:=1+(r_int+i) mod 2;
if (currency[q]=open)and
  ((size_of(curr_q[q])+size_of(curr_c[q]))<shortest) then
  begin
    best:=q;
    shortest:=size_of(curr_q[q])+size_of(curr_c[q])
  end
end;
set_attribute(element,3,best);
end;

```

```

procedure shortest_business(element:entity);
{** see which business_q to join - at least 1 counter **}
{** must be open **}
var
  i,r_int,best,shortest,q:integer;
begin
  best:=0;
  shortest:=9999;
  r_int:=trunc(2*rnd(12));
  for i:=1 to 2 do
    begin
      q:=1+(r_int+i) mod 2;
      if (business[q]=open)and
        ((size_of(bus_q[q])+size_of(bus_c[q]))<shortest) then
        begin
          best:=q;
          shortest:=size_of(bus_q[q])+size_of(bus_c[q])
        end
      end;
    set_attribute(element,3,best);
  end;
end;

```

```

procedure special_leaves;
{** special customer leaves - set states to finish scenario **}
{** called by service decision **}
begin
  special_in:=false;
  display_on:=false;
  clock_delay:=0;
  branch_open:=false;
end;

```

```

procedure service_decision(element:entity);
{** moves element onto the correct service **}
begin
  if attribute(element,2)=2 then {** transaction service **}
    begin
      move_on(element,decide,to_trans);
      schedule(5,0,element)
    end
  else if attribute(element,2)=1 then {** info service **}
    move_on(element,decide,info_q)
  else if attribute(element,2)=4 then {** currency service **}
    begin
      move_on(element,decide,to_currency);
    end
  end;
end;

```



```

    schedule(9,0,element)
end
else if attribute(element,2)=3 then {** business service **}
begin
    move_on(element,decide,to_business);
    schedule(12,0,element)
end
else if attribute(element,2)=0 then {** leave **}
begin
    dec(num_cust);
    move_on(element,decide,indoor);
    open_door;
    move_on(element,indoor,outdoor);
    close_door;
    case attribute(element,1) of
        2:move_on(element,outdoor,trans_cust);
        1:move_on(element,outdoor,info_cust);
        4:move_on(element,outdoor,curr_cust);
        3:move_on(element,outdoor,bus_cust);
        5:begin
            move_on(element,outdoor,special);
            special_leaves;
        end {**special**}
    end {**case**}
end {**else-if**}
end;

```

```

{*****}
{** B PHASE ** B PHASE ** B PHASE ** B PHASE ** B PHASE **}

```

```

procedure clock;
{** update 24 hour clock time **}
begin

    inc(mins);
    if mins=60 then
        begin
            mins:=0;
            inc(hours)
        end;
    if hours=24 then
        hours:=0;
        display_clock_time;
        if not shut then
            schedule(99,1,clock_dummy);
            delay(clock_delay)
        end;
    end;

```

```

procedure transaction_arrives(element:entity);
{** transaction customer arrives **}
begin
    if thinning(2) then
        begin
            inc(num_cust);
            set_attribute(element.1.2): {transaction customer}
            set_attribute(element.2.2): {do transaction next}
        end;
    end;

```

```

    move_on(element,trans_cust,outdoor);
    open_door;
    move_on(element,outdoor,indoor);
    close_door;
    move_on(element,indoor,decide);
    service_decision(element);
end;
if branch_open and not setup_stage then
    schedule(1,negexp(60/max_rate[2],1),head_of(trans_cust))
end;

{ **** }

procedure information_arrives(element:entity);
{** information customer arrives **}
begin
    if thinning(1) then
        begin
            inc(num_cust);
            set_attribute(element,1,1); {info customer}
            set_attribute(element,2,1); {do info next}
            move_on(element,info_cust,outdoor);
            open_door;
            move_on(element,outdoor,indoor);
            close_door;
            move_on(element,indoor,decide);
            service_decision(element);
        end;
    if branch_open and not setup_stage then
        schedule(2,negexp(60/max_rate[1],2),head_of(info_cust))
    end;

    { **** }

procedure currency_arrives(element:entity);
{** currency customer arrives **}
begin
    if thinning(4) then
        begin
            inc(num_cust);
            set_attribute(element,1,4); {currency customer}
            set_attribute(element,2,4); {do currency next}
            move_on(element,curr_cust,outdoor);
            open_door;
            move_on(element,outdoor,indoor);
            close_door;
            move_on(element,indoor,decide);
            service_decision(element);
        end;
    if branch_open and not setup_stage then
        schedule(3,negexp(60/max_rate[4],3),head_of(curr_cust))
    end;

    { **** }

procedure business_arrives(element:entity);
{** business customer arrives **}
begin
    if thinning(3) then

```

```

begin
  inc(num_cust);
  set_attribute(element,1,3); {business customer}
  set_attribute(element,2,3); {do business next}
  move_on(element,bus_cust,outdoor);
  open_door;
  move_on(element,outdoor,indoor);
  close_door;
  move_on(element,indoor,decide);
  service_decision(element);
end;
if branch_open and not setup_stage then
  schedule(4,negexp(60/max_rate[3],4),head_of(bus_cust))
end;

{*****}
{** special_arrives is among the queue selection routines **}
{*****}
procedure join_transaction(element:entity);
{** join best transactions queue **}
var
  q:integer;
begin
  if attribute(element,1) <> 5 then {** 5 is user decision **}
    shortest_transactions(element);
  q:=attribute(element,3);
  if q<>0 then
    move_on(element,to_trans,trans_q[q])
  else {** all transaction counter closed **}
    begin
      move_on(element,to_trans,decide);
      set_attribute(element,2,0);
      service_decision(element);
    end
  end;
end;

{*****}

procedure transaction_ends(element:entity);
{** transaction finishes, customer walks away from counter **}
var
  i,q:integer;
  shuffled:boolean;
begin
  q:=attribute(element,3);
  move_on(element,trans_c[q],to_trans);
  schedule(7,pos_normal(0.3,0.04,8),element);
end;

procedure decide_after_trans(element:entity);
{** after transaction service, choose other service or leave **}
var
  r:real;
begin
  move_on(element,to_trans,decide);
  if attribute(element,1)=2 then {** transaction cust **}
    begin
      r:=rnd(9);

```



```

    if r<0.02 then
        set_attribute(element,2,1)  {** needs information **}
    else
        set_attribute(element,2,0)  {** leave **}
    end
else
    set_attribute(element,2,0);      {** all others leave **}
service_decision(element);
end;

```

```

procedure information_ends(element:entity);
{** customer finished at information desk, decide on next service **}
{** or to leave **}
var
    q:integer;
    r:real;
begin
    q:=attribute(element,3);
    move_on(element,info_c[q],decide);
    {** work_out where customer will go next **}
    if attribute(element,1)=1 then {** info first customer **}
        begin
            r:=rnd(10);
            if r<0.5 then
                set_attribute(element,2,0) {** leave **}
            else if r<0.75 then
                set_attribute(element,2,2) {** transaction **}
            else if r<0.85 then
                set_attribute(element,2,3) {** currency **}
            else
                set_attribute(element,2,4) {** business **}
            end
        else {** not info first **}
            set_attribute(element,2,0); {** leave **}
        service_decision(element)
    end;
end;

```

```

procedure join_currency(element:entity);
{** join best currency queue **}
var
    q:integer;
begin
    if attribute(element,1)<>5 then {** 5 is user decision **}
        shortest_currency(element);
    q:=attribute(element,3);
    if q<>0 then
        move_on(element,to_currency,curr_q[q])
    else {** all currency counter closed **}
        begin
            move_on(element,to_currency,decide);
            set_attribute(element,2,0);
            service_decision(element);
        end
    end;
end;

```

```

procedure currency_ends(element:entity);

```



```

{** currency customer finished, move away from counter **}
var
  q:integer;
begin
  q:=attribute(element,3);
  move_on(element,curr_c[q],to_currency);
  schedule(11,pos_normal(0.1,0.015,13),element);
end;

```

```

procedure leave_after_currency(element:entity);
{** leave bank after currency service **}
begin
  set_attribute(element,2,0); {** leave **}
  move_on(element,to_currency,decide);
  service_decision(element)
end;

```

```

procedure join_business(element:entity);
{** choose to join best business queue **}
var
  q:integer;
begin
  if attribute(element,1)<>5 then {** 5 is user decision **}
    shortest_business(element);
  q:=attribute(element,3);
  if q <>0 then
    move_on(element,to_business,bus_q[q])
  else {** all business counters closed **}
    begin
      move_on(element,to_business,decide);
      set_attribute(element,2,0); {** leave bank **}
      service_decision(element);
    end
  end;
end;

```

```

procedure business_ends(element:entity);
{** business service ends, move away from counter **}
var
  q:integer;
begin
  q:=attribute(element,3);
  move_on(element,bus_c[q],to_business);
  schedule(14,pos_normal(0.1,0.015,14),element);
end;

```

```

procedure leave_after_business(element:entity);
{** leave bank after business service **}
begin
  set_attribute(element,2,0); {** leave **}
  move_on(element,to_business,decide);
  service_decision(element)
end;

```

```

{*****}
{** BRANCH OPEN / CLOSING & THINNING SCHEDULED EVENTS *****}

```

```

procedure next_period(element:entity);

```

```
{** controls moving onto next thinning period **}
```

```
var
```

```
  i:integer;
```

```
  p:integer;
```

```
begin
```

```
  p:=attribute(element,1);
```

```
  for i:= 1 to 4 do
```

```
    arr[i]:=arr_rate[i,p];
```

```
  inc(p);
```

```
  set_attribute(element,1,p);
```

```
  if (branch_open) and (p<30) then
```

```
    schedule(100,15,element)
```

```
  else
```

```
    branch_open:=false;
```

```
end;
```

```
{*****}
```

```
{** C PHASE ** C PHASE ** C PHASE ** C PHASE ** C PHASE **}
```

```
procedure transaction_free;
```

```
{** check for free counter at transaction service **}
```

```
var
```

```
  i:integer;
```

```
begin
```

```
{** note- if counter shut, there will be no queue **}
```

```
{** a counter which will shut must first serve the customers in queue **}
```

```
  for i:=1 to 5 do
```

```
    if (size_of(trans_q[i])>0) and (size_of(trans_c[i])=0) then
```

```
      begin
```

```
        element:=head_of(trans_q[i]);
```

```
        move_on(element,trans_q[i],trans_c[i]);
```

```
        schedule(6,gamma(4,0.5,6),element);
```

```
        if attribute(element,1)=5 then {** special **}
```

```
          begin {** record queueing time, switch off timer **}
```

```
            current_time:=timer_value(element);
```

```
            timing:=false;
```

```
            display_scores;
```

```
            delay_time:=0;
```

```
          {** record time that special is served **}
```

```
            record_q_data(current_special);
```

```
            q_data[amt_q_dat,11]:=attribute(element,3);
```

```
            timer_off(element);
```

```
          end;
```

```
          {** record queue data about special customer if they are in branch **}
```

```
          {** queueing for this counter type, and right queue has changed **}
```

```
          if special_in then
```

```
            if (attribute(current_special,2)=2)
```

```
              and (attribute(current_special,3)=i) then
```

```
                get_q_data;
```

```
          end
```

```
end;
```

```
procedure information_free;
```

```
{** check if an information counter is free **}
```

```
var
```

```
  i:integer;
```

```

element:entity;
service_available:boolean;
begin
  service_available:=false;
  for i:=1 to 4 do
    if (info[i]=open) then
      begin
        service_available:=true;
        if (size_of(info_q)>0) and (size_of(info_c[i])=0) then
          begin
            element:=head_of(info_q);
            set_attribute(element,3,i);
            move_on(element,info_q,info_c[i]);
            schedule(8,0.5+gamma(2.29,1.75,10),element);
            if attribute(element,1)=5 then  {** special **}
              begin
                {** record time in queue, switch off timer **}
                current_time:=timer_value(element);
                timing:=false;
                display_scores;
                delay_time:=0;
                {** record time that special is served **}
                record_q_data(current_special);
                q_data[amt_q_dat,11]:=attribute(element,3);
                timer_off(element);
              end;
            end;

            {** record queue data about special customer if they are in branch **}
            {** queueing for this counter type, and right queue has changed **}
            if special_in then
              if attribute(current_special,2)=1 then
                get_q_data;
              end
            end;
          end;
        end;
      end;
    if not service_available then  {** all info desks shut **}
      while size_of(info_q)>0 do
        begin
          element:=head_of(info_q);
          set_attribute(element,2,0);
          move_on(element,info_q,decide);
          service_decision(element)
        end
      end;
    end;

procedure currency_free;
{** check if currency counter is free **}
var
  i:integer;
begin
  for i:=1 to 2 do
    {** note- if a counter is shut, there will be no queue. If a counter **}
    {** is in the process of being closed, all customers in queue must be **}
    {** served first **}
    if (size_of(curr_q[i])>0) and (size_of(curr_c[i])=0) then
      begin
        element:=head_of(curr_q[i]);
        move_on(element,curr_q[i],curr_c[i]);
      end;
    end;
  end;

```



```

    schedule(10,0.5+gamma(2.78,0.9,7),element);
    if attribute(element,1)=5 then  {** special **}
        begin
            {** record time in queue, switch off timer **}
            current_time:=timer_value(element);
            timing:=false;
            display_scores;
            delay_time:=0;
            {** record time that specials is served **}
            record_q_data(current_special);
            q_data[amt_q_dat,11]:=attribute(element,3);
            timer_off(element);
        end;

{** record queue data about special customer if they are in branch **}
{** queueing for this counter type, and right queue has changed **}
    if special_in then
        if (attribute(current_special,2)=4)
            and (attribute(current_special,3) =i) then
            get_q_data;
        end
    end;

procedure business_free;
{** check if business counter is free **}
var
    i:integer;
begin
    for i:=1 to 2 do
        if (size_of(bus_q[i])>0) and (size_of(bus_c[i])=0) then
            {** note- if a counter is shut, there will be no queue. If a counter **}
            {** is in the process of being closed, all customers in queue must be **}
            {** served first **}
            begin
                element:=head_of(bus_q[i]);
                move_on(element,bus_q[i],bus_c[i]);
                schedule(13,gamma(3,0.75,15),element);
                if attribute(element,1)=5 then  {** special **}
                    begin
                        {** record time in queue, switch off timer **}
                        current_time:=timer_value(element);
                        timing:=false;
                        display_scores;
                        delay_time:=0;
                        {** record time that special is served **}
                        record_q_data(current_special);
                        q_data[amt_q_dat,11]:=attribute(element,3);
                        timer_off(element);
                    end;

                    {** record queue data about special customer if they are in branch **}
                    {** queueing for this counter type, and right queue has changed **}
                    if special_in then
                        if (attribute(current_special,2)=3)
                            and (attribute(current_special,3) =i) then
                            get_q_data;
                        end
                    end;
                end;
            end;
        end;
    end;
end;

```

```

procedure branch_empty;
{** checks when branch is empty after being closed **}
begin
  if not branch_open and not shut then
    if num_cust=0 then
      begin
        shut:=true;  {** if empty then can be considered truly shut **}
        schedule(202,100,branch);
      end
    end;
end;

procedure update_scores;
{** update current time in queue for special customer **}
begin
  if timing then
    begin
      current_time:=timer_value(current_special);
      display_scores
    end
  end;
end;

{** END OF C-PHASE *** END OF C-PHASE *** END OF C-PHASE **}

```

```

{***** USER INTERACTION WITH COUNTERS *****}

```

```

{** IDENTIFYING / HIGHLIGHTING / MOVING HIGHLIGHT FOR COUNTERS **}

```

```

procedure counter_func(c:integer;var f,p:integer);
{** for the purposes of controlling user interaction with counters for **}
{** setting staff - counters are number from 1 to 13 **}
{** convert counter number into function no (1 to 4) and position **}
begin
  if c<=4 then
    begin
      f:=1;
      p:=c
    end
  else if c<=9 then
    begin
      f:=2;
      p:=c-4
    end
  else if c<=11 then
    begin
      f:=3;
      p:=12-c
    end
  else
    begin
      f:=4;
      p:=14-c
    end
  end;
end;

```

```

procedure counter_loc(f,p:integer;var c:integer);

```

```

{** convert counter function and position to location in circle (1 to 13) **}
begin
  case f of
    1: c:=p;
    2: c:=4+p;
    3: c:=12-p;
    4: c:=14-p
  end
end;

```

```

procedure draw_outline(sc,x1,y1,x2,y2:integer;col:colour);
{** draw box round selected counter **}
begin
  draw_line(sc,x1,y1,x1,y2,col);
  draw_line(sc,x1,y2,x2,y2,col);
  draw_line(sc,x2,y2,x2,y1,col);
  draw_line(sc,x2,y1,x1,y1,col)
end;

```

```

procedure hi_info(c:integer;col:colour);
{** give location of a single info counter position **}
var
  x1,y1,x2,y2:integer;
begin
  x1:=218;y1:=114;x2:=252;y2:=142;
  draw_outline(1,x1+(c-1)*50,y1,x2+(c-1)*50,y2,col)
end;

```

```

procedure hi_trans(c:integer;col:colour);
{** give location of box round selected transaction counter **}
var
  x1,y1,x2,y2:integer;
begin
  x1:=518;y1:=118;x2:=550;y2:=152;
  draw_outline(1,x1,y1+(c-5)*40,x2,y2+(c-5)*40,col);
  draw_line(1,530,y1+(c-5)*40,530,y1+(c-5)*40,lightgray);
  draw_line(1,530,y2+(c-5)*40,530,y2+(c-5)*40,lightgray);
end;

```

```

procedure hi_bus(c:integer;col:colour);
{** give location of box round selected business counter **}
var
  x1,y1,x2,y2:integer;
begin
  x1:=418;y1:=318;x2:=452;y2:=351;
  draw_outline(1,x1-(c-10)*40,y1,x2-(c-10)*40,y2,col);
  draw_line(1,x1-(c-10)*40,330,x1-(c-10)*40,330,lightgray);
  draw_line(1,x2-(c-10)*40,330,x2-(c-10)*40,330,lightgray);
end;

```

```

procedure hi_curr(c:integer;col:colour);
{** give location of box round selected currency counter **}
var
  x1,y1,x2,y2:integer;
begin
  x1:=228;y1:=318;x2:=262;y2:=351;
  draw_outline(1,x1-(c-12)*40,y1,x2-(c-12)*40,y2,col);

```



```

draw_line(1,x1-(c-12)*40,330,x1-(c-12)*40,330,lightgray);
draw_line(1,x2-(c-12)*40,330,x2-(c-12)*40,330,lightgray);
end;

```

```

procedure hi_exit(forc,bak:colour);
{** draw an exit arrow to indicate choice of customer leaving bank **}
begin
  draw_triangle(1,80,210,90,195,90,225,forc,bak);

  draw_line(1,93,200,100,200,forc);
  draw_line(1,93,200,93,220,forc);
  draw_line(1,100,200,100,220,forc);
  draw_line(1,93,220,100,220,forc);
  fill_in(1,95,210,forc,bak);

  draw_line(1,103,200,108,200,forc);
  draw_line(1,103,200,103,220,forc);
  draw_line(1,108,200,108,220,forc);
  draw_line(1,103,220,108,220,forc);
  fill_in(1,105,210,forc,bak);

  draw_line(1,111,200,114,200,forc);
  draw_line(1,111,200,111,220,forc);
  draw_line(1,114,200,114,220,forc);
  draw_line(1,111,220,114,220,forc);
  fill_in(1,112,210,forc,bak);
end;

```

```

procedure hi_all_info(col:colour);
{** draw box round all information counters - this is used in customer **}
{** join/balk decision - select all counters since there is only a single **}
{** queue for information counters **}
begin
  draw_line(1,218,114,218,142,col);
  draw_line(1,218,114,403,114,col);
  draw_line(1,403,114,403,142,col);
  draw_line(1,218,142,403,142,col);
end;

```

```

procedure highlight(c:integer;col:colour);
{** determine which counter to highlight given counter position 1 - 13 **}
begin
  if (c>=1) and (c<=4) then
    hi_info(c,col)
  else if (c>=5) and (c<=9) then
    hi_trans(c,col)
  else if (c>=10) and (c<=11) then
    hi_bus(c,col)
  else if (c>=12) and (c<=13) then
    hi_curr(c,col)
end;

```

```

procedure unhighlight(c:integer);
{** determine which highlight to remove (draw over in black) given **}
{** counter position 1 - 13 **}
begin
  if (c>=1) and (c<=4) then
    hi_info(c,black)

```



```

else if (c>=5) and (c<=9) then
  hi_trans(c,black)
else if (c>=10) and (c<=11) then
  hi_bus(c,black)
else if (c>=12) and (c<=13) then
  hi_curr(c,black)
end;

```

```

function check_status(c:integer):status;
{** check open/closed/transition status of a counter when using **}
{** counter position 1 -13 **}
var
  s:status;
begin
  if (c>=1) and (c<=4) then
    s:=info[c]
  else if (c>=5) and (c<=9) then
    s:=transactions[c-4]
  else if (c>=10) and (c<=11) then
    s:=business[12-c]
  else if (c>=12) and (c<=13) then
    s:=currency[14-c]
  else
    s:=error;
  check_status:=s
end;

```

```

{***** CUSTOMER QUEUE JOINING ROUTINES *****}

```

```

procedure select_info_q(var q:integer;change:integer);
{** user selection of info queue or exit **}
var
  i:integer;
  service:boolean;
begin
  if q=0 then
    hi_exit(darkgray,black)
  else
    hi_all_info(black);
    q:=q+change;

    if q<0 then
      q:=1
    else if q>1 then
      q:=0;
    service:=false;
    for i:=1 to 4 do
      if info[i]=open then
        service:=true;
    if not service then
      q:=0;
    case q of
      0: hi_exit(lightred,green);
      1: hi_all_info(lightgreen);
    end
  end;

```

```

procedure select_trans_q(var q:integer;change:integer);
{** user selection of a transactions queue or exit **}
var
  service:boolean;
begin
  if q=0 then
    hi_exit(darkgray,black)
  else
    unhighlight(4+q);
  service:=false;
  q:=q+change;
  if q<0 then
    q:=5
  else if q>5 then
    q:=0;
  while (q<>0) and not service do
    begin
      if transactions[q]=open then
        service:=true
      else
        begin
          q:=q+change;
          if q<0 then
            q:=5
          else if q>5 then
            q:=0
        end
      end;
    if q=0 then
      hi_exit(lightred,green)
    else
      highlight(4+q,lightgreen)
    end;
  end;

```

```

procedure select_bus_q(var q:integer;change:integer);
{** user selection of a business queue or exit **}
var
  service:boolean;
begin
  if q=0 then
    hi_exit(darkgray,black)
  else
    unhighlight(12-q);
  service:=false;
  q:=q+change;
  if q<0 then
    q:=2
  else if q>2 then
    q:=0;
  while (q<>0) and not service do
    begin
      if business[q]=open then
        service:=true
      else
        begin
          q:=q+change;
          if q<0 then

```

```

        q:=2
    else if q>2 then
        q:=0
    end
end;
if q=0 then
    hi_exit(lightred,green)
else
    highlight(12-q,lightgreen)
end;

```

```

procedure select_curr_q(var q:integer;change:integer);
{** user selection of a currency queue or exit **}

```

```

var
    service:boolean;
begin
    if q=0 then
        hi_exit(darkgray,black)
    else
        unhighlight(14-q);
    service:=false;
    q:=q+change;
    if q<0 then
        q:=2
    else if q>2 then
        q:=0;
    while (q<>0) and not service do
        begin
            if currency[q]=open then
                service:=true
            else
                begin
                    q:=q+change;
                    if q<0 then
                        q:=2
                    else if q>2 then
                        q:=0
                    end
                end
            end;
        if q=0 then
            hi_exit(lightred,green)
        else
            highlight(14-q,lightgreen)
        end;
    end;

```

```

procedure queue_select(element:entity);
{** select appropriate queue for joining upon arrival at bank **}

```

```

var
    s1:string;
    counter:integer;
    q:integer;
    change:integer;
    finished:boolean;
    ch:char;
begin
    finished:=false;
    counter:=attribute(element,2);
    case counter of

```

```

1 : s1:='INFORMATION';
2 : s1:='TRANSACTIONS';
3 : s1:='BUSINESS';
4 : s1:='CURRENCY'
end;
write_message(s1,'Use <- -> to select a queue or leave bank',
              'Press RETURN to indicate final selection',yellow);

q:=0;
hi_exit(lightred,green);
{** move highlight onto first applicable queue - prevents chance of **}
{** people leaving bank by mistake **}
case counter of
  1:select_info_q(q,1);
  2:select_trans_q(q,1);
  3:select_bus_q(q,1);
  4:select_curr_q(q,1);
end;
repeat
  ch:=readkey;
  if ord(ch)=13 then {** return **}
  begin
    if q=0 then
      begin
        set_attribute(element,2,0);
        balk:=true;
        current_time:=timer_value(element);
        timer_off(element);
        timing:=false;
      end
    else
      set_attribute(element,3,q);
      finished:=true
    end
  else if ch= #0 then {** extended key **}
  begin
    ch:=readkey;
    case ord(ch) of
      75: change:=-1;
      77: change:=1;
      else change:=0;
    end;
    case counter of
      1:select_info_q(q,change);
      2:select_trans_q(q,change);
      3:select_bus_q(q,change);
      4:select_curr_q(q,change);
    end;
  end
until finished;
hi_exit(black,black);
case counter of
  1: hi_all_info(black);
  2: unhighlight(4+q);
  3: unhighlight(12-q);
  4: unhighlight(14-q)
end;
clear_message

```

end;

```
{*****}
{** RECORDING DECISION DATA **}
```

```
function customer_set(f,p:integer):sets;
{** used to convert from service number (1-4) **}
{** and queue number to a specific customer counter set **}
begin
  case f of
    1:customer_set:=info_c[p];
    2:customer_set:=trans_c[p];
    3:customer_set:=bus_c[p];
    4:customer_set:=curr_c[p]
  end
end;
```

```
function queue_set(f,p:integer):sets;
{** used to convert from service number (1-4) and queue number to a **}
{** specific counter queue set **}
begin
  case f of
    1:queue_set:=info_q;
    2:queue_set:=trans_q[p];
    3:queue_set:=bus_q[p];
    4:queue_set:=curr_q[p]
  end
end;
```

```
function counter_set(f,p:integer):sets;
{** Uses counter service number and position number to return the **}
{** appropriate server set **}
begin
  case f of
    1:counter_set:=info_s[p];
    2:counter_set:=trans_s[p];
    3:counter_set:=bus_s[p];
    4:counter_set:=curr_s[p]
  end
end;
```

```
procedure record_arrival_data(element:entity);
{** record state of bank upon arrival for selected service **}
var
  i:integer;
begin
  arrive_data[1]:=attribute(element,2); {** service required **}
  arrive_data[2]:=0; {** initialise number of counters open **}
  for i:=3 to 7 do
    arrive_data[i]:=-1; {** initialise counters as closed **}
  if arrive_data[1]=1 then {** info **}
  begin
    for i:=1 to 4 do
      if info[i]=open then
        begin
          inc(arrive_data[2]);
```



```

        arrive_data[i+2]:=size_of(info_c[i])+size_of(info_q)
    end
end
else if arrive_data[1]=2 then {** trans **}
begin
    for i:=1 to 5 do
        if transactions[i]=open then
            begin
                inc(arrive_data[2]);
                arrive_data[i+2]:=size_of(trans_c[i])+size_of(trans_q[i])
            end
        end
    end
else if arrive_data[1]=3 then {** business **}
begin
    for i:=1 to 2 do
        if business[i]=open then
            begin
                inc(arrive_data[2]);
                arrive_data[i+2]:=size_of(bus_c[i])+size_of(bus_q[i])
            end
        end
    end
else if arrive_data[1]=4 then {** currency **}
begin
    for i:=1 to 2 do
        if currency[i]=open then
            begin
                inc(arrive_data[2]);
                arrive_data[i+2]:=size_of(curr_c[i])+size_of(curr_q[i])
            end;
        end
    end
end;
end;

```

```

procedure record_q_data(element:entity);
{** record data about service while customer is in queue **}
var
    i:integer;
    counter,posit:integer;
begin
    if amt_q_dat<800 then
        begin
            inc(amt_q_dat);
            counter:=attribute(element,2);
            posit:=attribute(element,3);
            q_data[amt_q_dat,1]:=counter; {** service required **}
            q_data[amt_q_dat,2]:=0; {** initialise number of counters open **}
            for i:=3 to 7 do
                q_data[amt_q_dat,i]:=-1; {** initialise counters as closed **}
            end
            if counter=1 then {** info **}
                begin
                    for i:=1 to 4 do
                        if info[i]=open then
                            begin
                                q_data[amt_q_dat,2]:=q_data[amt_q_dat,2]+1;
                                q_data[amt_q_dat,i+2]:=size_of(info_c[i])+size_of(info_q)
                            end
                        end
                    end
                end
            else if counter=2 then {** trans **}
                begin

```

```

    for i:=1 to 5 do
        if transactions[i]=open then
            begin
                q_data[amt_q_dat,2]:=q_data[amt_q_dat,2]+1;
                q_data[amt_q_dat,i+2]:=size_of(trans_c[i])+size_of(trans_q[i])
            end
        end
    else if counter=3 then {** business **}
        begin
            for i:=1 to 2 do
                if business[i]=open then
                    begin
                        q_data[amt_q_dat,2]:=q_data[amt_q_dat,2]+1;
                        q_data[amt_q_dat,i+2]:=size_of(bus_c[i])+size_of(bus_q[i])
                    end
                end
            end
        else if counter=4 then {** currency **}
            begin
                for i:=1 to 2 do
                    if currency[i]=open then
                        begin
                            q_data[amt_q_dat,2]:=q_data[amt_q_dat,2]+1;
                            q_data[amt_q_dat,i+2]:=size_of(curr_c[i])+size_of(curr_q[i])
                        end
                    end
                end;
            {** account for fact that element may have moved queues : **}

            q_data[amt_q_dat,8]:=posit;
            q_data[amt_q_dat,9]:=position(element,queue_set(counter,posit));
            q_data[amt_q_dat,10]:=timer_value(element);

            end
        else {** run out of array space **}
            text_form(1,50,100,'OUT OF ARRAY SPACE',0,2,red,black)
        end;
end;

```

```

procedure get_q_data;
{** control collection of data about service while customer is queueing **}
var
    element:entity;
begin
    if special_in then
        begin
            element:=current_special;
            if (position(element,queue_set(arrive_data[1],attribute(element,3)))<>0) then
                begin
                    record_q_data(current_special);
                    q_data[amt_q_dat,11]:=attribute(element,3);
                {** collect data every minute. even if no queue swap has taken place **}
                end
            end
        end;
end;

```

```

procedure change_q(element:entity);
{** user controlled queue change - called if user presses space **}
var

```



```

old_q,new_q:integer;
counter:integer;
begin
if special_in then  {** special cust in system **}
begin
counter:=attribute(element,2);
old_q:=attribute(element,3);
if position(element,queue_set(counter,old_q))<>0 then
{** element is still in queue **}
begin
record_q_data(element); {** record state before change **}
queue_select(element);
if attribute(element,2)=0 then  {** balk **}
begin
move_on(element,queue_set(counter,old_q),decide);
service_decision(element);
q_data[amt_q_dat,11]:=0
end
else
begin
new_q:=attribute(element,3);
q_data[amt_q_dat,11]:=new_q; {** record change **}
if new_q<>old_q then  {** swap queues **}
move_on(element,queue_set(counter,old_q),
queue_set(counter,new_q))
end;
end {if still in q}
end
end;

{*****}
{** SPECIAL CUSTOMER ARRIVES **}

procedure special_arrives(element:entity);
{** arrival routine for a special - user controlled - customer **}
{** This is a scheduled B-phase routine, but calls data capture routines **}
var
decision:integer;
s:string;
begin
inc(num_cust);
display_on:=true;
clock_delay:=500;
delay_time:=2;
form_screens;
text_form(1,140,400,'Exit Penalty ',0,2,yellow,lightgray);
str(balk_penalty:2:0,s);
text_form(1,346,400,s,0,2,lightred,lightgray);
text_form(1,378,400,' Minutes',0,2,yellow,lightgray);

current_special:=element; {** to find element in change_q **}
set_attribute(element,1,5); {** special **}
special_in:=true;
{** type of customer specified in custf file **}
set_attribute(element,2,spec_type);
case spec_type of
1: change_description(element,'I',lightred,white);
2: change_description(element,'T',lightred,white);

```

```

3: change_description(element,'B',lightred,white);
4: change_description(element,'C',lightred,white);
end;
move_on(element,special,outdoor);
open_door;
move_on(element,outdoor,indoor);
close_door;
move_on(element,indoor,decide);
timer_on(element);
timing:=true;
record_arrival_data(element);
{** records state of bank in arrival_data array **}

{** generate customer arrival decision - balk or join a particular queue **}
queue_select(element);

{*** queue_select(element); ***}
{**record final decision **}
if attribute(element,2)=0 then {** has balked **}
  arrive_data[8]:=0
else
  arrive_data[8]:=attribute(element,3);
service_decision(element);
end;

{*****}
{** SIMULATION LOGIC CONTROL ENGINE **}

procedure run_simulation;
begin
  while simulating and scenario do
    begin
      {A phase}
      advance(next_event,time,element);

      {B phase}
      case next_event of
        -1      : change_q(current_special);
        0 {interact   } : own_interaction;
        1 {transaction cust} : transaction_arrives(element);
        2      : information_arrives(element);
        3      : currency_arrives(element);
        4      : business_arrives(element);
        5 {move to trans Q} : join_transaction(element);
        6 {}      : transaction_ends(element);
        7      : decide_after_trans(element);
        8      : information_ends(element);
        9      : join_currency(element);
        10     : currency_ends(element);
        11     : leave_after_currency(element);
        12     : join_business(element);
        13     : business_ends(element);
        14     : leave_after_business(element);
        30     : special_arrives(element);
        99     : clock;
        100    : next_period(element);
        201    : set_scenario;
        202    : end_scenario;

```

```

203         : start_scenario;
300         : get_q_data;
end;

```

```
{C phase}
```

```

transaction_free;
information_free;
currency_free;
business_free;
branch_empty;
update_scores;
end;
end;
{*****}

```

```

procedure end_simulation;
var ch:string;
begin
    open_own;
    text_form(1,80,30,'END OF SIMULATION',0,1,red,lightgray);
    text_form(1,90,50,'PLEASE WAIT',0,1,red,lightgray);
    ch:=readkey;
    ch:=readkey;
    close_own;
    closegraph;
end;

```

```

{*****}
{** MAIN PROGRAM **}

```

```

begin
    setup;
    define_entities;
    load_arrival_rates;
    session;
    delay_time:=2; {** simkit variable controlling sim speed **}
    form_screens;
    run_simulation;
    close(scenef);
    close(arr_df);
    close(q_df);
    end_simulation
end.

```

Appendix C

Bank Simulation Scenarios

The scenarios created for the data collection stage of the Bank simulation study were stored in files and the appropriate one loaded in for a particular data collection session.

An original set of scenarios were prepared for the pilot study, and then subsequently altered slightly for the full study. Each study used two different scenario files. In the full study, only one of these was seen by any individual participant. The files contain just the numbers, but headings are included here for convenience.

Pilot Study - Scene1.inp

Cust. Type	Entry Time	No. Customers				Counters open (1) and Closed (0)												
		I	T	B	C	Transactions					Information				Bus		Curr	
1	396	9	3	1	0	1	1	1	1	1	1	1	0	1	1	1	1	
1	137	5	2	3	1	1	0	0	0	0	1	0	0	0	1	0	1	1
1	31	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0
3	265	4	2	7	2	1	1	0	1	0	1	0	0	0	1	1	1	1
3	124	0	0	3	0	1	1	1	1	0	1	0	0	0	1	0	1	0
1	284	11	10	3	2	1	1	1	1	1	1	1	1	1	0	1	1	1
4	125	5	6	2	8	1	1	1	1	1	1	1	1	1	1	1	1	1
2	147	5	20	2	1	0	1	1	0	1	1	0	1	1	0	1	0	1
4	3	3	2	0	3	0	1	1	1	0	0	0	1	0	0	1	1	1
2	326	1	7	0	1	0	0	0	1	1	1	0	1	1	1	1	1	0
4	164	1	15	2	12	1	0	0	1	1	1	1	1	1	1	1	1	1
2	117	6	8	1	1	0	1	1	1	1	1	0	1	0	1	0	0	1
1	119	3	0	1	1	1	0	0	0	0	1	1	1	0	1	1	1	1
4	106	9	0	3	6	1	1	1	1	0	1	0	0	1	1	1	1	0
3	152	7	0	4	2	0	1	0	1	0	0	1	1	1	1	1	1	1
2	232	0	12	2	7	1	1	1	0	1	0	1	1	1	1	1	1	1
2	313	7	7	3	3	0	1	1	0	0	0	0	0	1	1	0	0	1
1	320	10	6	1	7	1	1	0	1	1	1	1	1	1	1	1	1	1
2	336	2	4	1	0	0	1	0	0	0	1	0	0	0	1	1	1	1
2	389	1	14	5	6	0	1	0	0	1	1	1	1	1	1	1	1	1
2	14	7	3	1	5	1	1	1	1	0	0	1	0	0	1	1	1	1
2	351	6	8	1	0	1	1	1	1	0	1	1	0	0	1	1	1	1
2	356	0	6	3	7	0	0	0	1	0	1	0	1	0	1	0	1	1
3	333	11	5	9	0	1	1	1	0	1	0	0	0	1	1	1	1	1
3	47	12	17	2	0	1	1	1	1	1	1	1	1	1	1	1	1	0
2	77	15	1	2	7	1	1	1	1	0	0	0	0	1	1	0	1	1
1	188	4	10	4	4	1	1	1	1	1	1	1	1	0	1	1	1	1
4	265	3	3	2	10	0	1	0	0	0	1	1	1	0	0	1	1	1
1	285	7	3	0	3	1	0	0	0	0	0	0	0	1	1	0	1	1

Customer Type : 1=Information, 2=Transactions, 3=Business, 4=Currency

Pilot Study - Scene2.inp

Cust. Type	Entry Time	No. Customers				Counters open (1) and Closed (0)															
		I	T	B	C	Transactions					Information				Bus		Curr				
1	385	7	10	2	3	1	1	1	1	1	1	1	1	1	1	0	1	0			
2	126	8	15	2	7	1	0	1	1	1	1	1	1	0	0	1	1	1			
2	214	4	4	3	3	0	1	1	1	0	0	0	1	1	1	1	0	1			
2	362	14	15	7	7	1	1	1	1	1	0	1	0	1	1	1	1	1			
1	310	5	6	4	5	0	1	0	1	0	0	1	1	1	1	1	1	1			
2	391	4	10	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1			
3	160	1	11	1	2	0	1	1	1	0	0	1	1	1	1	0	1	0			
3	98	3	4	2	4	1	0	0	1	1	0	1	1	0	1	0	1	1			
1	161	2	3	2	3	1	0	1	0	0	0	0	1	0	1	1	0	1			
1	282	8	2	2	2	1	1	0	1	0	0	0	0	1	1	0	1	0			
1	50	3	4	1	0	1	0	1	0	0	1	1	1	0	1	0	1	0			
3	252	6	3	3	1	1	1	0	0	1	0	0	0	0	1	1	1	0			
3	299	2	2	11	2	1	0	0	0	1	0	0	0	0	1	1	1	0			
2	237	1	10	7	1	0	0	0	1	0	0	0	1	1	1	1	1	0			
1	288	4	15	2	1	1	0	1	1	1	0	1	1	1	0	1	1	1			
2	391	6	6	1	0	1	0	1	1	0	1	0	1	1	1	0	1	0			
3	123	9	6	5	4	1	1	1	0	1	0	0	1	1	1	1	0	1			
1	160	7	1	3	5	1	0	0	1	1	1	0	0	0	1	0	1	1			
2	356	1	5	4	0	0	0	1	1	0	0	0	0	1	1	1	0	1			
4	68	7	5	2	6	1	1	0	0	0	1	0	1	0	1	0	1	1			
2	256	9	19	2	1	0	1	1	1	1	1	1	1	1	0	1	1	1			
4	307	2	7	1	4	1	0	0	0	1	0	0	0	1	1	1	1	0			
4	120	6	4	5	5	1	1	0	0	1	0	0	0	1	1	1	1	1			
3	378	3	10	5	2	1	1	1	1	0	0	1	1	1	1	0	1	1			
2	326	2	24	2	4	0	0	0	1	1	1	1	1	1	1	1	1	1			
4	202	3	4	2	3	1	0	0	0	0	0	1	1	0	1	0	0	1			
1	189	6	5	2	0	1	1	1	0	1	1	1	1	0	1	0	1	0			
2	307	3	12	0	3	0	0	1	1	0	1	1	0	1	0	1	0	1			
4	147	3	3	4	2	0	1	0	0	1	0	0	1	1	1	1	0	1			

Customer Type : 1=Information, 2=Transactions, 3=Business, 4=Currency

Main Study - Scene1.inp

Cust. Type	Entry Time	No. Customers				Counters open (1) and Closed (0)												
		I	T	B	C	Transactions					Information				Bus		Curr	
1	137	5	2	3	1	1	0	0	0	0	1	0	0	0	1	0	1	1
1	31	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0
3	124	0	0	4	0	1	1	1	1	0	1	0	0	0	1	0	1	0
1	284	14	10	3	2	1	1	1	1	1	1	1	1	1	0	1	1	1
4	125	5	6	2	8	1	1	1	1	1	1	1	1	1	1	1	1	1
2	147	5	30	2	1	0	1	1	0	1	1	0	1	1	0	1	0	1
3	333	11	5	12	0	1	1	1	0	1	0	0	0	1	1	1	1	1
4	3	3	2	0	3	0	1	1	1	0	0	0	1	0	0	1	1	1
2	326	1	9	0	1	0	0	0	1	1	1	0	1	1	1	1	1	0
4	164	1	15	2	12	1	0	0	1	1	1	1	1	1	1	1	1	1
2	117	6	9	1	1	0	1	1	1	1	1	0	1	0	1	0	0	1
1	119	3	0	1	1	1	0	0	0	0	1	1	1	0	1	1	1	1
4	106	9	0	3	6	1	1	1	1	0	1	0	0	1	1	1	1	0
3	152	7	0	4	2	0	1	0	1	0	0	1	1	1	1	1	1	1
2	232	0	12	2	7	1	1	1	0	1	0	1	1	1	1	1	1	1
2	313	7	9	3	3	0	1	1	0	0	0	0	0	1	1	0	0	1
1	320	10	6	1	7	1	1	0	1	1	1	1	1	1	1	1	1	1
2	336	2	5	1	0	0	1	0	0	0	1	0	0	0	1	1	1	1
1	396	10	3	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1
2	389	1	16	5	6	0	1	0	0	1	1	1	1	1	1	1	1	1
2	14	7	3	1	5	1	1	1	1	0	0	1	0	0	1	1	1	1
3	265	4	2	9	2	1	1	0	1	0	1	0	0	0	1	1	1	1
2	351	6	10	1	0	1	1	1	1	0	1	1	0	0	1	1	1	1
2	356	0	7	3	7	0	0	0	1	0	1	0	1	0	1	0	1	1
3	47	12	17	2	0	1	1	1	1	1	1	1	1	1	1	1	1	0
2	77	15	1	2	7	1	1	1	1	0	0	0	0	1	1	0	1	1
1	188	5	10	4	4	1	1	1	1	1	1	1	1	0	1	1	1	1
4	265	3	3	2	10	0	1	0	0	0	1	1	1	0	0	1	1	1
1	285	7	3	0	3	1	0	0	0	0	0	0	0	1	1	0	1	1

Customer Type : 1=Information, 2=Transactions, 3=Business, 4=Currency

Main Study - Scene2.inp

Cust. Type	Entry Time	No. Customers				Counters open (1) and Closed (0)															
		I	T	B	C	Transactions					Information				Bus		Curr				
1	385	7	10	2	3	1	1	1	1	1	1	1	1	1	0	1	0	1	0		
2	126	8	20	2	7	1	0	1	1	1	1	1	1	0	0	1	1	1	1		
3	160	1	11	1	2	0	1	1	1	0	0	1	1	1	1	0	1	0	0		
2	362	14	21	7	7	1	1	1	1	1	0	1	0	1	1	1	1	1	1		
1	310	7	6	4	5	0	1	0	1	0	0	1	1	1	1	1	1	1	1		
2	391	4	10	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1		
3	98	3	4	2	4	1	0	0	1	1	0	1	1	0	1	0	1	1	1		
1	161	3	3	2	3	1	0	1	0	0	0	0	1	0	1	1	0	1	1		
4	120	6	4	5	5	1	1	0	0	1	0	0	0	1	1	1	1	1	1		
1	282	8	2	2	2	1	1	0	1	0	0	0	0	1	1	0	1	0	0		
1	50	5	4	1	0	1	0	1	0	0	1	1	1	0	1	0	1	0	0		
3	252	6	4	3	1	1	1	0	0	1	0	0	0	0	1	1	1	0	0		
3	299	2	2	16	2	1	0	0	0	1	0	0	0	0	1	1	1	0	0		
2	237	1	14	7	1	0	0	0	1	0	0	0	1	1	1	1	1	0	0		
1	288	4	15	2	1	1	0	1	1	1	0	1	1	1	0	1	1	1	1		
2	391	6	6	1	0	1	0	1	1	0	1	0	1	1	1	0	1	0	0		
1	160	9	1	3	5	1	0	0	1	1	1	0	0	0	1	0	1	1	1		
2	356	1	7	4	0	0	0	1	1	0	0	0	0	1	1	1	0	1	1		
4	68	7	5	2	6	1	1	0	0	0	1	0	1	0	1	0	1	1	1		
2	256	9	24	2	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1		
4	307	2	7	1	4	1	0	0	0	1	0	0	0	1	1	1	1	0	0		
2	214	4	4	3	3	0	1	1	1	0	0	0	1	1	1	1	0	1	1		
3	378	3	10	8	2	1	1	1	1	0	0	1	1	1	1	0	1	1	1		
2	326	2	37	2	4	0	0	0	1	1	1	1	1	1	1	1	1	1	1		
4	202	3	4	2	3	1	0	0	0	0	0	1	1	0	1	0	0	1	1		
1	189	6	5	2	0	1	1	1	0	1	1	1	1	0	1	0	1	0	0		
2	307	3	13	0	3	0	0	1	1	0	1	1	0	1	0	1	0	1	1		
4	147	3	3	4	2	0	1	0	0	1	0	0	1	1	1	1	0	1	1		

Customer Type : 1=Information, 2=Transactions, 3=Business, 4=Currency

Appendix D

Bank Simulation - Participants Instructions

The following presents the instructions given to participants of the data collection stage for the Bank simulation study.

PARTICIPANTS INSTRUCTIONS

Aims of the Study

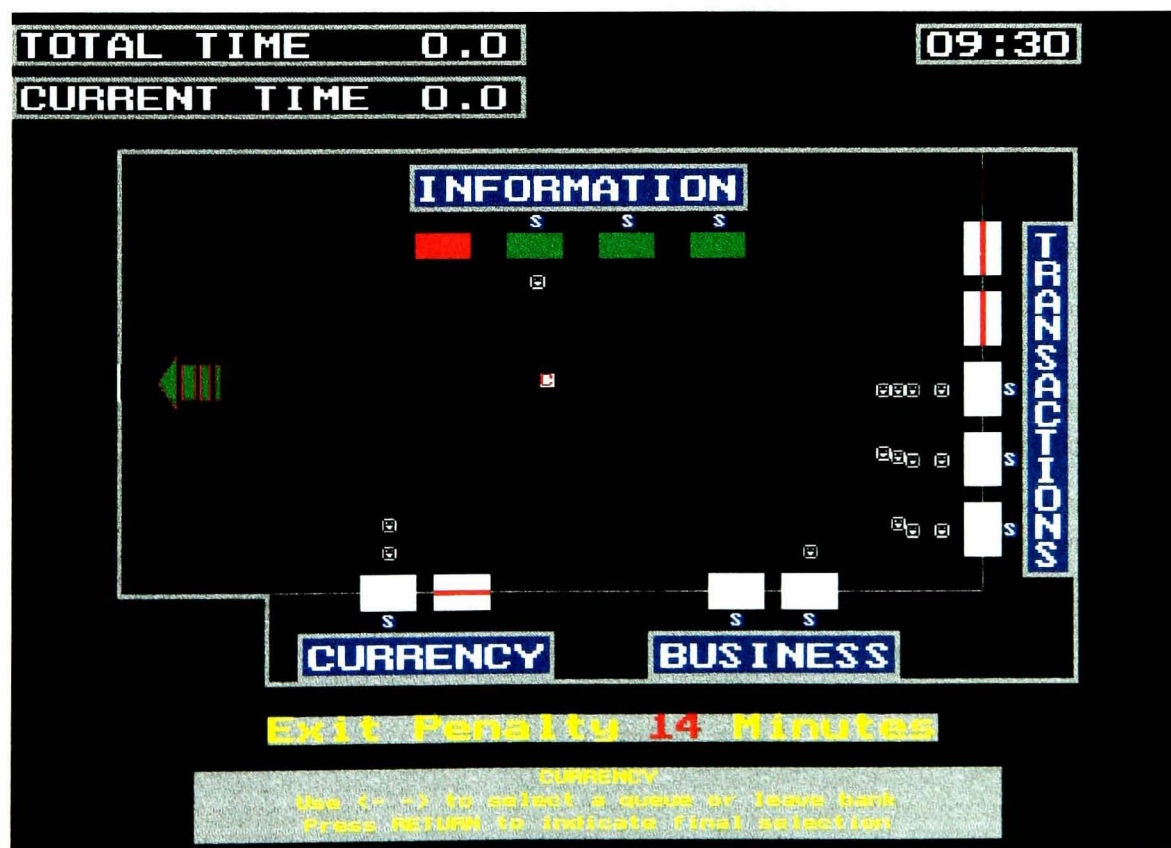
Thankyou for agreeing to take part in this study, your help is very much appreciated. The aim of the study is to replicate the behaviour of bank customers as they make decisions about queuing in a bank. To do this, data must be collected about the about the sorts of decisions which people make. This is where you come in.

You will be shown a set of scenarios and asked to make various decisions for each one. The details of the situation and your decisions will be recorded by the computer for analysis later.

The Scenarios

The simulation involves a large high street bank. There are four sorts of services available. The customer whom you control will display a letter indicating which service you require:

- (I) Information - personal banking, general inquiries.
- (T) Transactions - paying in, withdrawing cash etc.
- (B) Business - transaction services for businesses.
- (C) Currency - currency exchange services.



Your aim is to make decisions which will minimise the amount of time that the customers need to queue before receiving the service they require. The computer will keep track of how long you spend queuing for each scenario and display this time on the screen (Current Time), it will also keep a running count of the total time spent queuing during the scenarios (Total Time).

You will be shown a series of scenarios where each type of service will have one or more counters open. Except for Information where there is one queue only, the services will have a queue for each counter that is open. You must decide whether or not you will stay in the bank, and if so, which queue to join for the service that you require. While queuing you may also decide to swap queues if you think that another queue will be quicker.

For each scenario you will be shown an 'Exit Penalty'. This is a time penalty which will be charged for exiting the bank before the customer has received the service. Therefore if you think that the time spent queuing will be longer than the exit penalty, it is worth leaving the bank. You may leave the bank when you first arrive or at any time while you are queuing, however the exit penalty will be **added** to the amount of time you have spent in the bank on that particular scenario. For example, if the exit penalty is 10 minutes and you have been queuing for 5 minutes before you decide to leave, the total penalty incurred will be 15 minutes.

Instructions

1. *Arriving at the bank*

At the beginning of the scenario you will see the bank with customers queueing for various services. The customer you control will enter the bank. It will be different from the other customers in that it has a letter showing which service is required (the service will also be shown on the instruction panel at the bottom of the screen). To the left of the screen you will see an arrow pointing at the door - this is the option to exit the bank. Pressing the **left or right cursor** key (**←, →**) allows you to change this option to one of the counters of the relevant service. Keep pressing the cursor keys until you see the option you require. To select the option press **return** (**↵**). The customer will then join the appropriate queue or leave the bank.

2. *Changing queues or leaving bank*

At any time while in a queue you can swap queues or leave the bank by pressing the **space bar**. The counter you are queuing for will be highlighted. This can be changed by pressing the **left or right cursor** keys (**←, →**) and the final selection made using **return** (**↵**). You can leave the bank by selecting the large arrow pointing at the door. If you do leave you will receive the exit penalty on top of the time that you stayed in the bank during the scenario. Do not worry if you press the space bar and do not want to change queues since you can always opt to stay in the same queue without losing your place.

Before you start the recording session, you will be given a set of practice scenarios to enable you to get used to using the program, and also to give you some idea about how long each of the services takes. Just like real life, you find that different people take different amounts of time at the counter, but some types of service are inherently longer than others.

If you make a mistake, i.e. pressed a button when you did not mean to, make this known as soon as it happens. That record will then be removed from the later analysis.

The end of the session

At the end of the session you will be asked a few questions about your experiences with the game.

Appendix E

Example Decision Outcome Datasets

The Bank simulation decision collection framework stored data on the decisions that were made by participants for each scenario. Examples of the data saved for one participant are shown here. Separate data was stored for the initial decision on arrival, and for the decisions made once a queue had been joined. Table headings are added to the datasets for presentation purposes.

Arrival Decision

The arrival decision involves deciding whether or not to join a queue, and if so, which queue to join.

Exit Penalty	Service Required	No. Open Counters	Queue Sizes for Each Counter (-1 = closed or not available)					Queue Joined
5	1	1	5	-1	-1	-1	-1	0
8	1	1	-1	-1	1	-1	-1	1
14	3	1	4	-1	-1	-1	-1	1
14	1	4	11	11	11	11	-1	1
11	4	2	4	4	-1	-1	-1	1
5	2	4	7	8	-1	8	7	0
2	3	2	6	6	-1	-1	-1	0
2	4	2	2	1	-1	-1	-1	0
8	2	4	3	2	-1	2	2	5
14	4	2	6	6	-1	-1	-1	0
5	2	3	3	3	-1	2	-1	4
14	1	1	3	-1	-1	-1	-1	1
8	4	1	6	-1	-1	-1	-1	0
5	3	2	2	2	-1	-1	-1	2
5	2	4	3	-1	3	3	3	0
2	2	1	-1	-1	-1	-1	9	0
11	1	3	8	8	-1	8	-1	1
8	2	1	-1	5	-1	-1	-1	2
8	1	4	7	7	7	7	-1	1
14	2	5	3	3	3	4	2	5
14	2	1	-1	-1	3	-1	-1	3
11	3	2	5	4	-1	-1	-1	2
8	2	2	-1	5	5	-1	-1	0
8	2	2	-1	4	-1	3	-1	4
5	3	2	1	1	-1	-1	-1	1
5	2	1	-1	-1	-1	-1	1	5
2	1	4	2	2	2	2	-1	0
11	4	2	5	5	-1	-1	-1	2
11	1	1	7	-1	-1	-1	-1	0

The **exit penalty** is chosen based on the identity number of the participant and the time of arrival of the special customer (this is different for each scenario). Every 10th participant will see the same scenarios with the same exit penalties.

The **service required** shows which service the special customer needed to join for the scenario. The meanings of the values are : 1 = Information, 2 = Transactions, 3 = Business, 4 = Currency.

The **number of open counters** is for the required service only. This will be in the range 1..4 for Information, 1..5 for Transactions and 1..2 for Business and Currency.

The **queue size for each counter** shows the queue inclusive of the person who is being served. The values are in order for counters 1 to 5 for each service. A value of -1 indicates that the counter is closed, or that a counter for that queue position does not exist for the required service. Information has a single queue, but the total queue size (single queue plus person being served) is shown for each of the counters.

The **queue joined** shows the decision made for the customer. A decision value of 0 means that the customer left the bank, otherwise the queue number is given (Information always has a queue number of 1).

In-Queue Decision

The in-queue decision shows the activities of the customer once they have joined the queue. This involves swapping queues, reneging from the queue, or staying where they are. A series of data entries is made for each scenario where the customer stays in the bank. Data is recorded whenever the customer changes queues, or reneges, or when the queue shortens in length.

Exit Penalty	Service Reqd	Counters Open	Queue Sizes					Current Queue	Position in Queue	Time in Queue	New Queue
8	1	1	-1	-1	1	-1	-1	3	0	0.99	3
14	3	1	5	-1	-1	-1	-1	1	3	1.2	1
14	3	1	4	-1	-1	-1	-1	1	2	1.98	1
14	3	1	3	-1	-1	-1	-1	1	1	4.31	1
14	3	1	2	-1	-1	-1	-1	1	0	6.42	1
14	1	4	12	12	12	12	-1	1	10	1.01	1
14	1	4	11	11	11	11	-1	1	9	5.01	1
14	1	4	10	10	10	10	-1	1	8	5.4	1
14	1	4	9	9	9	9	-1	1	7	5.43	1
14	1	4	8	8	8	8	-1	1	6	6.32	1
14	1	4	7	7	7	7	-1	1	5	7.45	1
14	1	4	6	6	6	6	-1	1	4	8.43	1
14	1	4	5	5	5	5	-1	1	3	10.39	1
14	1	4	4	4	4	4	-1	1	2	12.92	1
14	1	4	3	3	3	3	-1	1	1	13.36	1
14	1	4	2	2	2	2	-1	2	0	13.5	2
11	4	2	4	4	-1	-1	-1	1	3	0.57	1
11	4	2	4	4	-1	-1	-1	1	2	2.35	1
11	4	2	3	3	-1	-1	-1	1	1	5.13	1
11	4	2	2	1	-1	-1	-1	1	0	13.21	1
8	2	4	3	1	-1	2	3	5	2	0.57	2
8	2	4	3	1	-1	2	2	2	0	0.62	2
5	2	3	3	3	-1	2	-1	4	1	1.32	4
5	2	3	3	3	-1	2	-1	4	0	1.74	4
14	1	1	4	-1	-1	-1	-1	1	2	1	1
14	1	1	3	-1	-1	-1	-1	1	1	2.6	1
14	1	1	2	-1	-1	-1	-1	1	0	3.9	1
5	3	2	3	2	-1	-1	-1	2	1	1.52	2
5	3	2	3	1	-1	-1	-1	2	0	1.96	2
5	2	4	3	-1	3	3	3	5	2	0.53	5
5	2	4	2	-1	3	2	2	5	1	1.64	5
5	2	4	2	-1	2	1	1	5	0	3.65	5
11	1	3	9	9	-1	9	-1	1	7	1.01	1
11	1	3	8	8	-1	8	-1	1	6	1.55	1
11	1	3	8	8	-1	8	-1	1	5	9.05	1
11	1	3	7	7	-1	7	-1	1	4	9.8	1
11	1	3	6	6	-1	6	-1	1	3	12.61	1
11	1	3	6	6	-1	6	-1	1	2	13.22	1
11	1	3	5	5	-1	5	-1	1	1	15.9	1
11	1	3	4	4	-1	4	-1	4	0	17.47	4
8	2	1	-1	5	-1	-1	-1	2	4	0.06	2
8	2	1	-1	5	-1	-1	-1	2	3	2.1	2
8	2	1	-1	7	-1	-1	-1	2	2	4.48	2
8	2	1	-1	7	-1	-1	-1	2	1	5.25	2
8	2	1	-1	7	-1	-1	-1	2	0	5.73	2
8	1	4	7	7	7	7	-1	1	6	0.97	1
8	1	4	6	6	6	6	-1	1	5	1.01	1
8	1	4	5	5	5	5	-1	1	4	2.29	1
8	1	4	4	4	4	4	-1	1	3	2.47	1
8	1	4	3	3	3	3	-1	1	2	4.79	1
8	1	4	2	2	2	2	-1	1	1	5.44	1
8	1	4	1	1	1	1	-1	4	0	6.84	4

Exit Penalty	Service Reqd	Counters Open	Queue Sizes						Current Queue	Position in Queue	Time in Queue	New Queue
14	2	5	2	2	2	2	2	5	1	2.11	5	
14	2	5	1	1	2	2	2	5	1	2.67	5	
14	2	5	2	1	1	1	1	5	0	5.59	5	
14	2	1	-1	-1	3	-1	-1	3	2	0.04	3	
14	2	1	-1	-1	3	-1	-1	3	1	1.25	3	
14	2	1	-1	-1	2	-1	-1	3	0	1.97	3	
11	3	2	4	4	-1	-1	-1	2	3	0.92	2	
11	3	2	4	3	-1	-1	-1	2	2	1.64	2	
11	3	2	3	2	-1	-1	-1	2	1	3.88	2	
11	3	2	3	1	-1	-1	-1	2	0	4.47	2	
8	2	2	-1	3	-1	3	-1	4	2	1.84	4	
8	2	2	-1	2	-1	2	-1	4	1	3.96	4	
8	2	2	-1	3	-1	3	-1	4	0	7.35	4	
5	3	2	2	0	-1	-1	-1	1	1	1.1	2	
5	3	2	1	1	-1	-1	-1	2	0	1.1	2	
5	2	1	-1	-1	-1	-1	1	5	0	0.04	5	
11	4	2	5	5	-1	-1	-1	2	4	0.57	2	
11	4	2	4	4	-1	-1	-1	2	3	3.84	2	
11	4	2	3	3	-1	-1	-1	2	2	6.41	2	
11	4	2	3	2	-1	-1	-1	2	1	7.44	2	
11	4	2	2	1	-1	-1	-1	2	0	10.02	2	

The first 4 columns of data show the same information as the arrival decision data.

The **current queue** shows the queue that the customer was in just prior to the in-queue decision.

The **position in queue** shows how close to the front of the queue the customer is. A position of 0 shows that service for the customer is about to commence.

The **time in queue** shows how long the customer has spent queuing since first starting to queue.

The **new queue** is the queue after the decision has been made. This might be the same as the current queue, in which case the customer has not moved, it might be a different queue, or it might be 0 indicating that the customer has reneged.

Appendix F

Bank Simulation Questionnaire

The following is the questionnaire that was used after the decision data collection for the Bank simulation.

Bank Customer Simulation Experiment Sheet

Subject Details

Subject No:

Date:

Scenario:

Balk Penalty:

Overall Score:

Contact Details:

Debrief

1. How difficult was it to judge whether you should stay in the branch or leave:

	EASY			HARD	
	1	2	3	4	5
OVERALL					
INFORMATION					
TRANSACTIONS					
BUSINESS					
CURRENCY					

2. Did you develop any specific rules in your head for deciding whether or not to stay in the bank?

<p>INFORMATION</p> <p>Rules:</p> <p>Stick with them?</p> <p>Why not?</p>
<p>TRANSACTIONS</p> <p>Rules:</p> <p>Stick with them?</p> <p>Why not?</p>
<p>BUSINESS</p> <p>Rules:</p> <p>Stick with them?</p> <p>Why not?</p>
<p>CURRENCY</p> <p>Rules:</p> <p>Stick with them?</p> <p>Why not?</p>

3. How often did you stay in the same queue once you had joined one?

Why?

4. What criteria did you use for deciding to change queues?

TRANSACTIONS
BUSINESS
CURRENCY

How often did you find that the decision to change paid off?

5. Did you ever leave the bank after joining a queue?

What factors lead you to that decision?

6. Overall, did you feel that your performance in the simulation:

got worse

stayed the same

got better

Why?

Appendix G

Decision Mapping Forms for Bank Simulation

The decision mapping forms are used to keep track of the decision models. The use of the Hybrid decision modelling approach means that the decision models may be made up of several sub-models, each of which needs to be linked together.

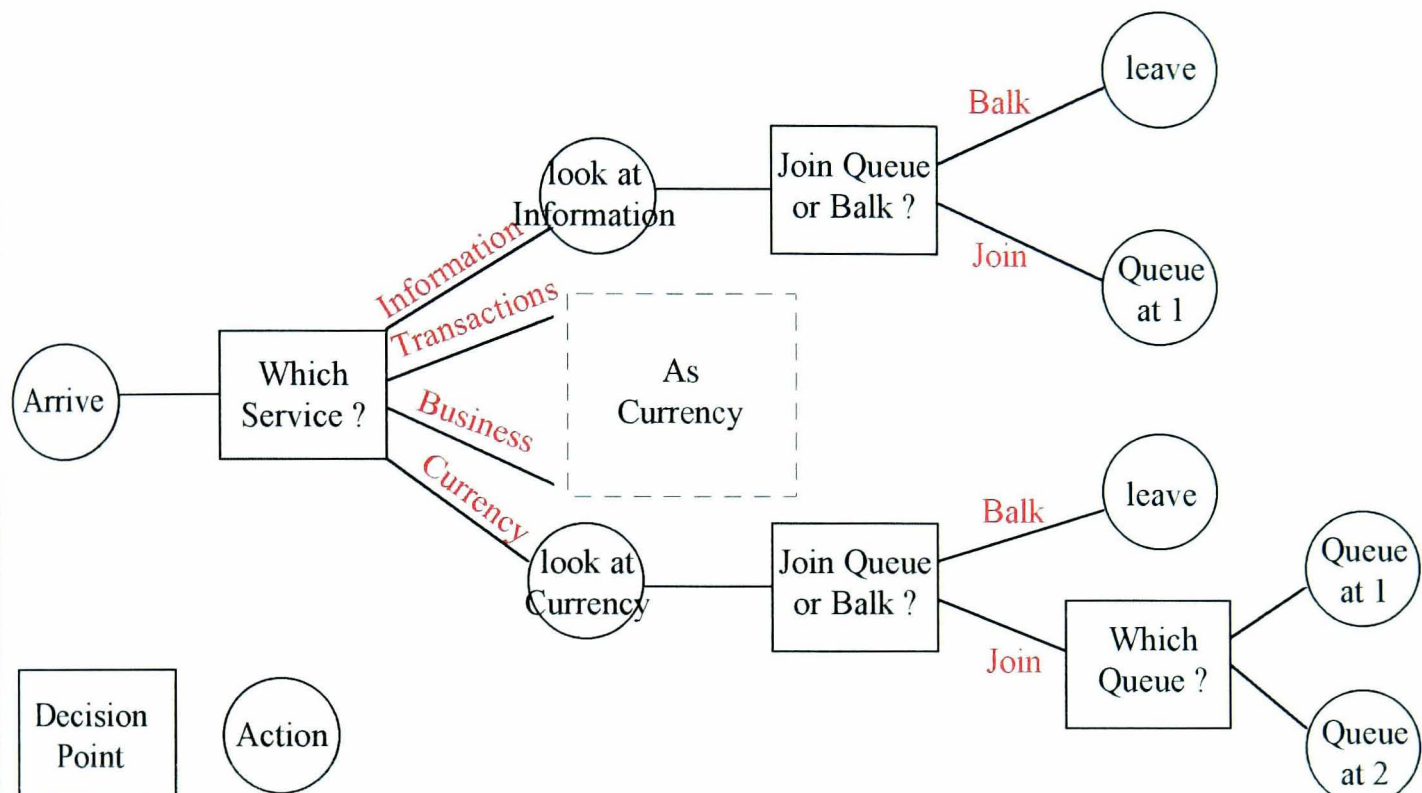
Each form has two parts. These are usually one side each of a double sided sheet, but have been separated into two sheets here.

The top of Form A contains a heading section that records the title of the project, the decision name, and the sub-decision name. The level is used to indicate where the sub-decision fits into the overall decision process. A hybrid model would usually have an overview sheet that shows all of the sub-processes fitted together, and then more detailed sheets for each of the sub-processes. This can be seen for the Arrival decision. The decision relation diagram allows the structure of decision to be displayed graphically in terms of the sub-decisions and the possible outcomes. The convention used was to represent a decision as a square, and an outcome as a circle. The description allows brief text to be added about the decision. The decision outcome allows the types of output variables to be stated, along with possible classifications or ranges. The decision criteria allows the input decisions to be listed.

Form B allows a statement of the modelling methods used. The model names and descriptions generally refer to the names of any files or programs created, with details on how they were created or how they are to be used. The section on neural network structures allows details of the variables used in the model, and how input values to be used in the network and outputs from the model should be scaled. The formal rules allows the rule-based components of the model to be specified.

Project: **BANK**Decision: **ARRIVAL**Level: **OVERVIEW**

Sub-Decision name: -

Decision Relation Diagram**Description****Overview of whole decision process.****See breakdown of decision into sub-decisions for higher detail.****Decision Outcome****Join a particular queue for desired service, or leave bank (balk).****Decision Criteria****Service Type****Number of counters for each service****Size of queue for each counter**

Modelling Method
N/A

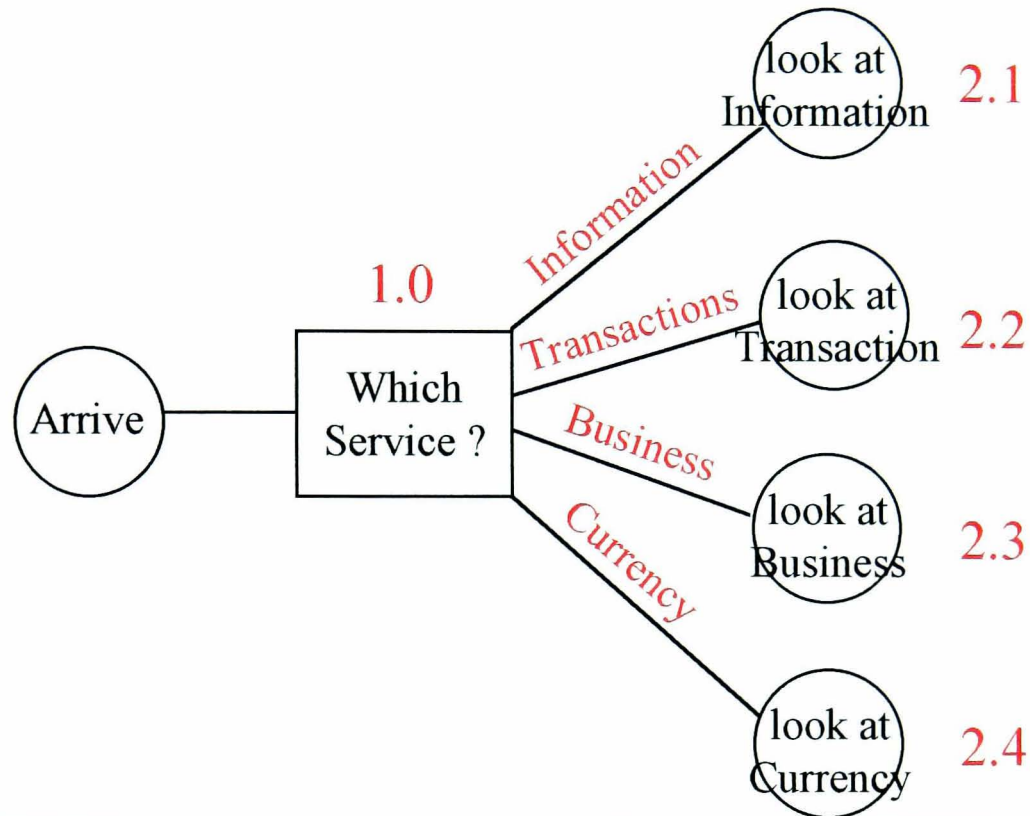
Model Names & Descriptions
N/A

Neural Network Structures
N/A

Formal Rules
N/A

Project: **BANK** Decision: **ARRIVAL** Level: **1.0**
 Sub-Decision name: **Which Service**

Decision Relation Diagram



Description

A decision based on the customer entity type. No internal criteria for decision. Generated by independent arrival processes - one for each service point. Each uses thinning to variable arrival rate during the day.

Special case is some Information customers who may go on to other services, the start of each new service can be treated as an arrival for decision making purposes.

Decision Outcome : Classification

- 1 : Information
- 2 : Transactions
- 3 : Business
- 4 : Currency

Decision Criteria

Type of customer

Modelling Method

Simple rules based on entity attribute

Model Names & Descriptions

N/A

Neural Network Structures

N/A

Formal Rules

If type_attribute = 1 then look_at_information

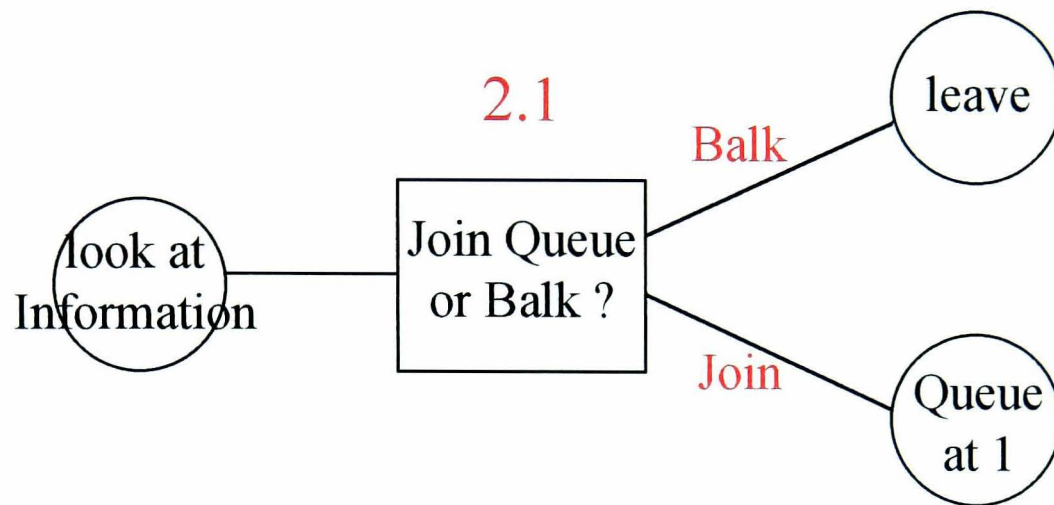
If type_attribute = 2 then look_at_transactions

If type_attribute = 3 then look_at_business

If type_attribute = 4 then look_at_currency

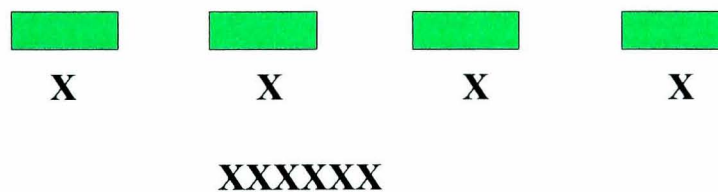
Project: BANK **Decision: ARRIVAL** **Level: 2.1**
Sub-Decision name: Information - Join or Balk

Decision Relation Diagram



Description

Information has between 1 and 4 counters open, with a single queue for all the counters.



Decision Outcome Classification

0 : Balk (Action = leave)

1 : Join Information Queue

Decision Criteria

Number of Counters open (1 - 4)

Queue Size (Number in single queue)

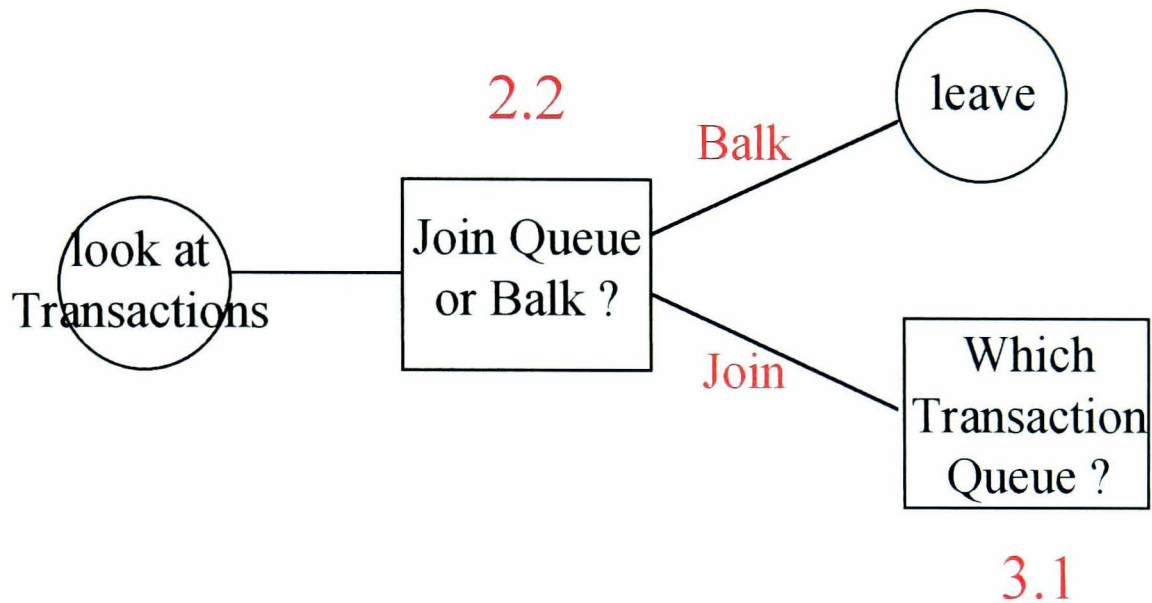
(Note original data included customer queuing, but training data does not.)

Network is only trained to deal with cases where all the counters are occupied.)

Modelling Method**Neural network & rules****Model Names & Descriptions****Network Files : INF_I2.NOD, INF_I2.WGT - INVAR MODEL****Training Data : INF_I.TRN****Testing Data : Using program TSTINVAR.PAS****Neural Network Structures****3 Inputs, 2 Hidden Layers, 2 Outputs****Input 1: Number Counters Open (scaled from 1 to 4, scaled to 0 to 1)****Input 2: Shortest Queue Length (scaled from 0 to 15, scaled to 0 to 1)****Input 3: Random Number Variable (range 0 to 1)****Output 1: Stay Outcome****Output 2: Balk Outcome****Decision by winner takes all (highest value) comparison of outcome****Formal Rules****If > 0 (counter open and free) then join nearest free counter to door (leftmost)****If 0 counters open then leave****else consult neural network model**

Project: BANK **Decision: ARRIVAL** **Level: 2.2**
Sub-Decision name: Transactions - Join or Balk

Decision Relation Diagram



Description

Transactions has between 1 and 5 counters open, with a separate queue for each counter. Decision is whether to join one of the queues or leave the bank. A further sub-decision is handled separately regarding which queue to join.

Decision Outcome Classification

0 : Balk (Action = leave)

1 : Join queue (Decision 3.1 which queue ?)

Decision Criteria

Number of counters open (1 - 5)

Length of shortest queue (integer)

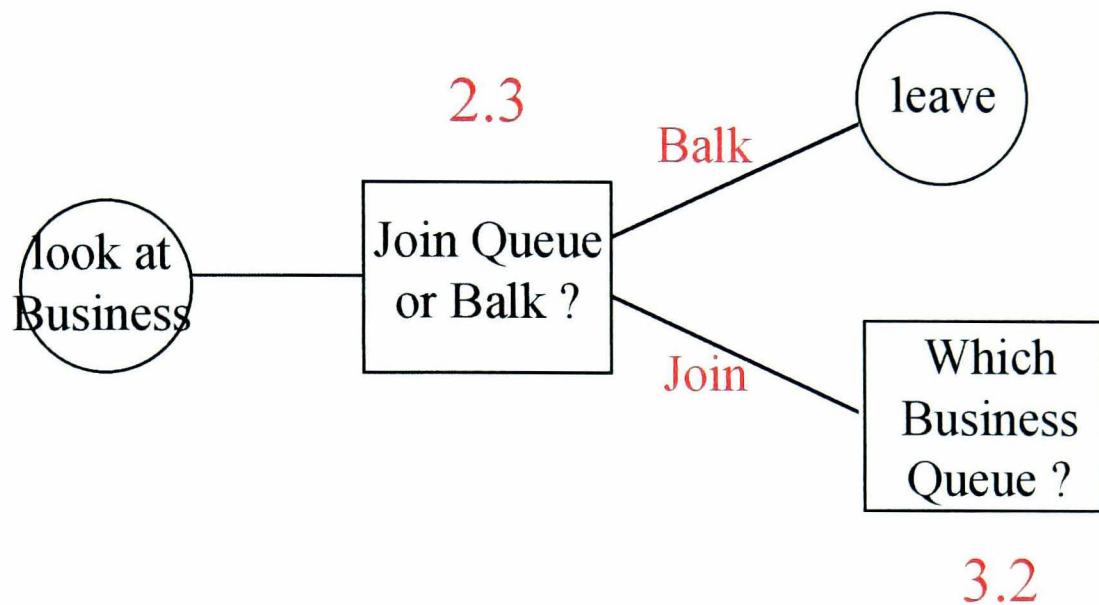
Modelling Method**Neural network and rules****Model Names & Descriptions****Network Files : TRA_IB2.NOD, TRA_IB2.WGT - INVAR MODEL****Training Data : TRANS_IB.TRN****Testing Data : Using program TSTINVAR.PAS**

Note: Training data contains an artificially generated data for the case with 1 counter open, and 10 people in the queue - all probability variable values show balking outputs (1 0).

Neural Network Structures**3 Inputs, 2 Hidden Layers, 2 Outputs****Input 1: Number Counters Open (scaled from 1 to 5, scaled to 0 to 1)****Input 2: Shortest Queue Length (scaled from 0 to 15, scaled to 0 to 1)****Input 3: Random Number Variable (range 0 to 1)****Output 1: Stay Outcome****Output 2: Balk Outcome****Decision by winner takes all comparison of outcome****Formal Rules****If >0 counters (open and free) then stay - decide on which queue to join****If 0 counters open then leave****else consult neural network model**

Project: **BANK** Decision: **ARRIVAL** Level: **2.3**
 Sub-Decision name: **Business - Join or Balk**

Decision Relation Diagram



Description

Business has either 1 or 2 counters open, each with a separate queue. Decision is whether to stay and the bank and wait for service or leave.
A further sub-decision for which queue to join is handled separately.

Decision Outcome Classification

0 : Balk (Action = leave)
1 : Join queue (Decision 3.2 which queue ?)

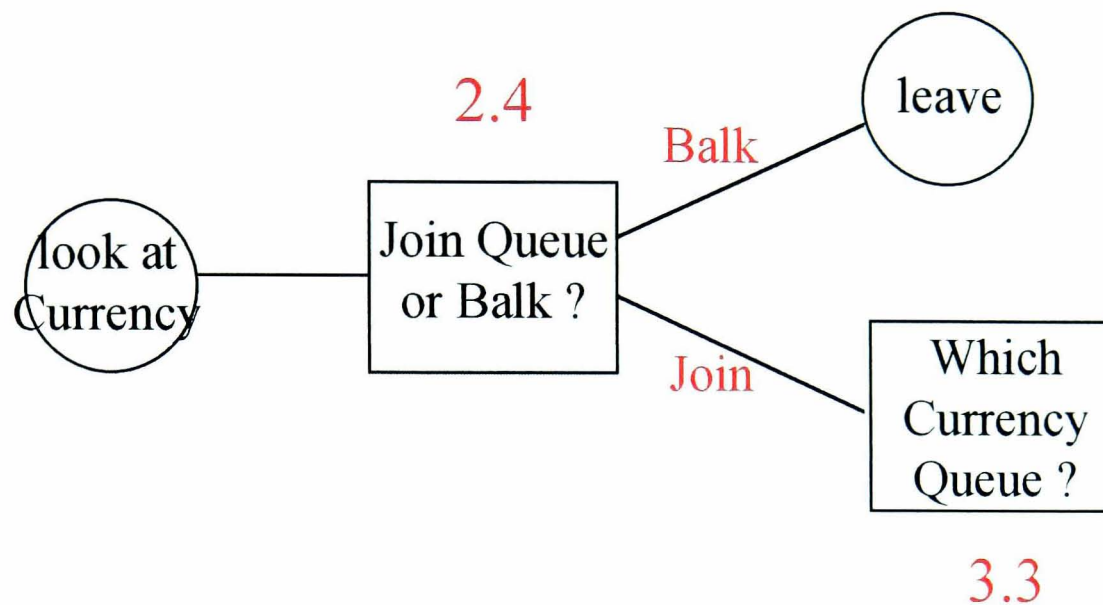
Decision Criteria

Number of counters open (1 - 2)
Length of shortest queue (integer)

Modelling Method**Neural network and rules****Model Names & Descriptions****Network Files : BUS_I3.NOD, BUS_I3.WGT - INVAR MODEL****Training Data : BUS_I.TRN****Testing Data : Using program TSTINVAR.PAS****Neural Network Structures****3 Inputs, 3 Hidden Layers, 2 Outputs****Input 1: Number Counters Open (scaled from 1 to 2, scaled to 0 to 1)****Input 2: Shortest Queue Length (scaled from 0 to 15, scaled to 0 to 1)****Input 3: Random Number Variable (range 0 to 1)****Output 1: Stay Outcome****Output 2: Balk Outcome****Decision by winner takes all comparison of outcome****Formal Rules****If > 0 counters (open and free) then decide which queue to join****If 0 counter open then leave****else consult neural network model**

Project: **BANK** Decision: **ARRIVAL** Level: **2.4**
 Sub-Decision name: **Currency - Join or Balk**

Decision Relation Diagram



Description

Currency has either 1 or 2 counters open, each with a separate queue. Decision is whether to stay and the bank and wait for service or leave.
 A further sub-decision for which queue to join is handled separately.

Decision Outcome Classification

- 0 : Balk (Action = leave)
- 1 : Join queue (Decision 3.3 which queue ?)

Decision Criteria

Number of counters open (1 - 2)
 Length of shortest queue (integer)

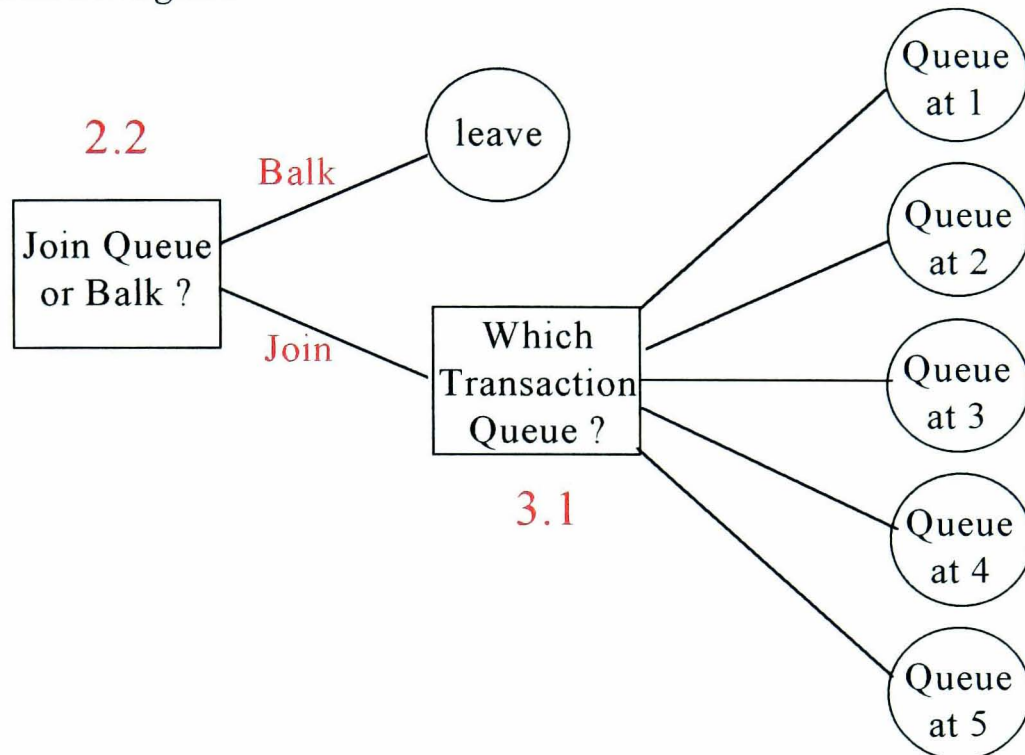
Modelling Method**Neural network and rules****Model Names & Descriptions****Network Files : CUR_IB2.NOD, CUR_IB2.WGT - INVAR MODEL****Training Data : CUR_IB.TRN****Testing Data : Using program TSTINVAR.PAS**

Note: Training data contains an artificially generated data for the cases with 1 counter and 2 counters open, and 8 people in the queue - all probability variable values show balking outputs (1 0).

Neural Network Structures**3 Inputs, 2 Hidden Layers, 2 Outputs****Input 1: Number Counters Open (scaled from 1 to 2, scaled to 0 to 1)****Input 2: Shortest Queue Length (scaled from 0 to 15, scaled to 0 to 1)****Input 3: Random Number Variable (range 0 to 1)****Output 1: Stay Outcome****Output 2: Balk Outcome****Decision by winner takes all comparison of outcome****Formal Rules****If > 0 counters (open and free) then decide which queue to join****If 0 counter open then leave****else consult neural network model**

Project: **BANK** Decision: **ARRIVAL** Level: **3.1**
 Sub-Decision name: **Join Transaction - Which Queue ?**

Decision Relation Diagram



Description

Between 1 and 5 counters open. The decision to stay has already been made. The decision is which individual counter queue to join.

Decision Outcome Classification

- 1 : counter 1
- 2 : counter 2
- 3 : counter 3
- 4 : counter 4
- 5 : counter 5

Decision Criteria

Which queues are the shortest
 Pattern of queue sizes

Modelling Method
Relative probabilities

Model Names & Descriptions

N/A

Neural Network Structures

N/A

Formal Rules

Preference weightings for queues:

Queue1	Queue2	Queue3	Queue4	Queue5
1.667	1.022	1.282	1.065	0.502

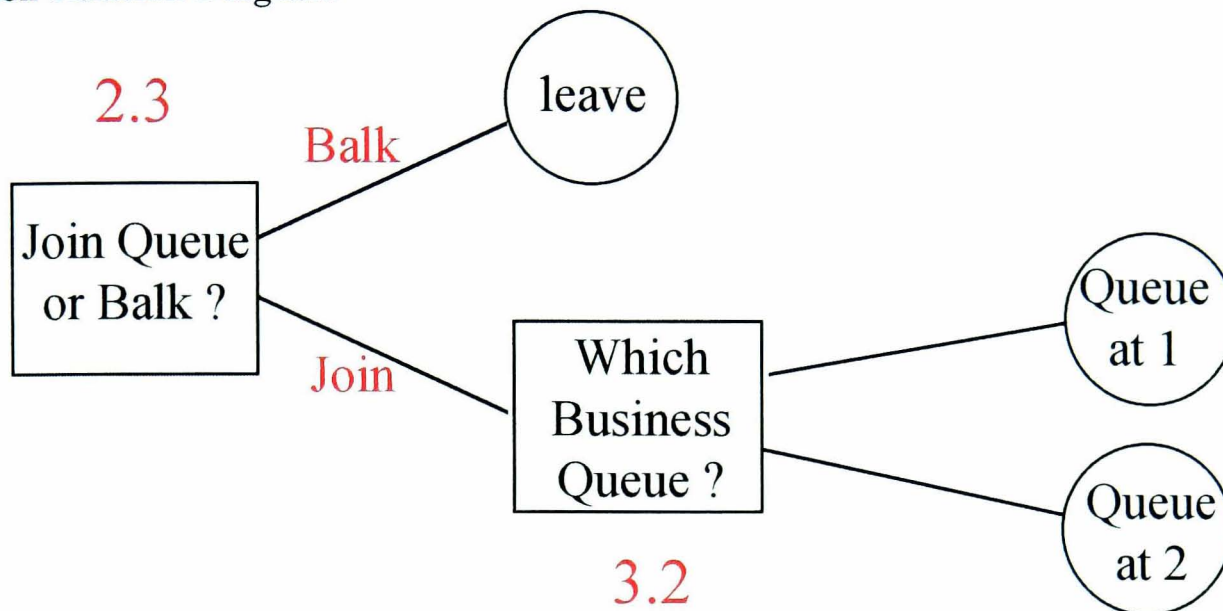
If only 1 shortest queue then join that
else join queue through random sampling using probabilities of :

$$\text{Prob}(\text{Queue } i) = \text{Weight}(\text{Queue } i) / \text{Sum of Weights of all Shortest Queues}$$

for any Shortest Queue i

Project: **BANK** Decision: **ARRIVAL** Level: **3.2**
 Sub-Decision name: **Join Business - Which Queue ?**

Decision Relation Diagram



Description

Either 1 or 2 counters open. The decision to stay has already been made. The decision is which individual counter queue to join.

Decision Outcome Classification

1 : counter 1
 2 : counter 2

Decision Criteria

Which counters open
 length of queue 1
 length of queue 2

Modelling Method**Rules / Sampling****Model Names & Descriptions**

N/A

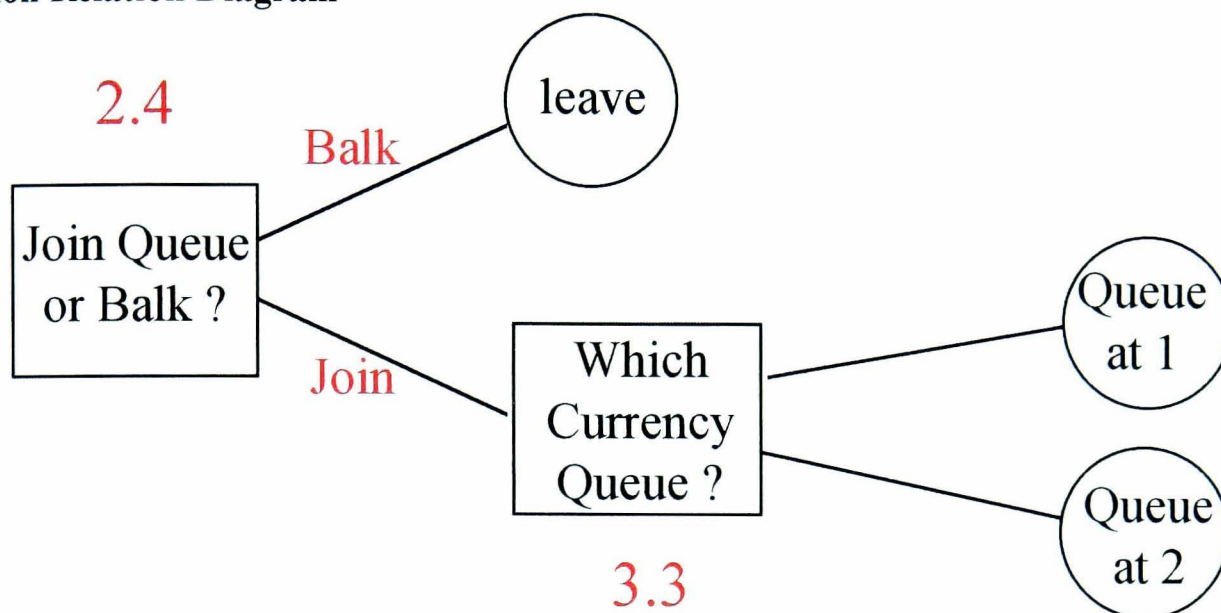
Neural Network Structures

N/A

Formal Rulesif only **Counter 1 open** then **join Queue 1**if only **Counter 2 open** then **join Queue 2**if only **1 queue shortest** then **join shortest queue**if **both queues shortest** then **join Queue 1 (prob 0.714) or Queue 2 (prob 0.286)**

Project: **BANK** Decision: **ARRIVAL** Level: **3.3**
 Sub-Decision name: **Join Currency - Which Queue ?**

Decision Relation Diagram



Description

Either 1 or 2 counters open. The decision to stay has already been made. The decision is which individual counter queue to join.

Decision Outcome Classification

1 : counter 1
 2 : counter 2

Decision Criteria

Which counters open
 length of queue 1
 length of queue 2

Modelling Method**Rules / Sampling****Model Names & Descriptions**

N/A

Neural Network Structures

N/A

Formal Rulesif only **Counter 1 open** then **join Queue 1**if only **Counter 2 open** then **join Queue 2**if only **1 queue shortest** then **join shortest queue**if both **queues shortest** then **join Queue 1 (prob 0.643) or Queue 2 (prob 0.357)**

Project: BANK **Decision: RENEGE**
Sub-Decision name: - All Services

Level: 1

Decision Relation Diagram

Description

Renege decision can occur after a customer has joined a service queue, until they are served. The decision is to stay in the queue and wait, or leave. This decision has a time dimension to it since there is no clear discrete point in time where the decision is required to take place.

Decision Outcome Classification

0 : Stay in Queue
1 : Renege

Decision Criteria

Position in queue

Average time per person served while customer has been in queue

Modelling Method**Rules and Neural Network****Model Names & Descriptions**

Network Files : RENEGE6.NOD, RENEGE6.WGT - OUTDATA MODEL

Training Data : RENEGE3.TRN

Testing Data : RENEGE3.TST

Neural Network Structures

2 Inputs, 6 Hidden Layers, 2 Outputs

**Input 1 : Queue Position Class = 0 if 3 or closer to front of queue (excl. person served)
= 1 if more than 3 from front of queue**

Input 2 : Average service rate in queue while customer queuing

Output 1 : Probability of Reneging

Output2 : Probability of Staying

Compare against fixed random number for customer

Formal Rules

If average service time > than max observed by customer then consider reneging

If queue position = 1 (next to be served) then stay in queue

If position class 0 and average service time < 4.0 then stay in queue

If position class 0 and average service time > 4.0 then consult network (Decision Grp B)

If position class 1 then consult network (Decision Grp A)

Only re-sample random number for customer when they newly enter decision Grp A or B and when average service time > max observed by customer.

Project: BANK**Decision: QUEUE SWAP****Level: 1****Sub-Decision name: - All Services****Decision Relation Diagram****Description**

Queue swap decision involves customers swapping between queues for the same service to improve their position in the queue. The decision has a time dimension, although this is ignored, and the decision is reviewed when a queue changes state.

Decision Outcome Classification

0 : Stay in Queue

n : Swap to Queue n

Decision Criteria

Position in queue

Modelling Method

Rules and Sampling

Model Names & Descriptions

N/A

Neural Network Structures

N/A

Formal Rules

If neighbouring queue (excluding closed counters) allows improvement in queue position then consider swapping

If customer tested for swapping then swap (prob. 91.67%)

If more than one neighbouring queue exists then one shall be selected at random to start testing

If the tested customer does not swap then next eligible customer will be tested. If another neighbouring queue exists, these will be tested in turn, otherwise the next eligible person is the next one back in the queue

If the opportunity exists to get served immediately by swapping queues then if no previous customer has changed:

If 1 neighbouring queue then customer at the end of the neighbouring queue shall swap.

If 2 neighbouring queues then first person tested who is at end of a queue shall swap.

If there is no-one in the neighbouring queues then customer in the next nearest queue shall swap.

Appendix H

Stay / Balk Decisions for the Bank Simulation

H1 : Information Service

The training data and models are presented for the Stay/Balk decisions for the Information service. The INVAR, OUTDATA and OUTNET models will be dealt with separately.

INVAR Models

The training data (before scaling) for the INVAR model approach is shown. Note that queue length is for the single queue and does not include the people being served.

Number Counters	Queue Length	Probability	Stay Decision	Balk Decision	Number Counters	Queue Length	Probability	Stay Decision	Balk Decision
1	0	0.05	1	0	3	1	0.05	1	0
1	0	0.15	1	0	3	1	0.15	1	0
1	0	0.25	1	0	3	1	0.25	1	0
1	0	0.35	1	0	3	1	0.35	1	0
1	0	0.45	1	0	3	1	0.45	1	0
1	0	0.55	1	0	3	1	0.55	1	0
1	0	0.65	1	0	3	1	0.65	1	0
1	0	0.75	1	0	3	1	0.75	1	0
1	0	0.85	1	0	3	1	0.85	1	0
1	0	0.95	1	0	3	1	0.95	1	0
1	2	0.05	1	0	3	3	0.05	1	0
1	2	0.15	1	0	3	3	0.15	1	0
1	2	0.25	1	0	3	3	0.25	1	0
1	2	0.35	1	0	3	3	0.35	1	0
1	2	0.45	1	0	3	3	0.45	1	0
1	2	0.55	1	0	3	3	0.55	1	0
1	2	0.65	1	0	3	3	0.65	1	0
1	2	0.75	0	1	3	3	0.75	1	0
1	2	0.85	0	1	3	3	0.85	0	1
1	2	0.95	0	1	3	3	0.95	0	1
1	4	0.05	1	0	3	5	0.05	1	0
1	4	0.15	1	0	3	5	0.15	1	0
1	4	0.25	0	1	3	5	0.25	1	0
1	4	0.35	0	1	3	5	0.35	0	1
1	4	0.45	0	1	3	5	0.45	0	1
1	4	0.55	0	1	3	5	0.55	0	1
1	4	0.65	0	1	3	5	0.65	0	1
1	4	0.75	0	1	3	5	0.75	0	1
1	4	0.85	0	1	3	5	0.85	0	1
1	4	0.95	0	1	3	5	0.95	0	1
1	6	0.05	0	1	3	7	0.05	1	0
1	6	0.15	0	1	3	7	0.15	1	0
1	6	0.25	0	1	3	7	0.25	1	0
1	6	0.35	0	1	3	7	0.35	1	0
1	6	0.45	0	1	3	7	0.45	1	0
1	6	0.55	0	1	3	7	0.55	1	0
1	6	0.65	0	1	3	7	0.65	0	1
1	6	0.75	0	1	3	7	0.75	0	1
1	6	0.85	0	1	3	7	0.85	0	1

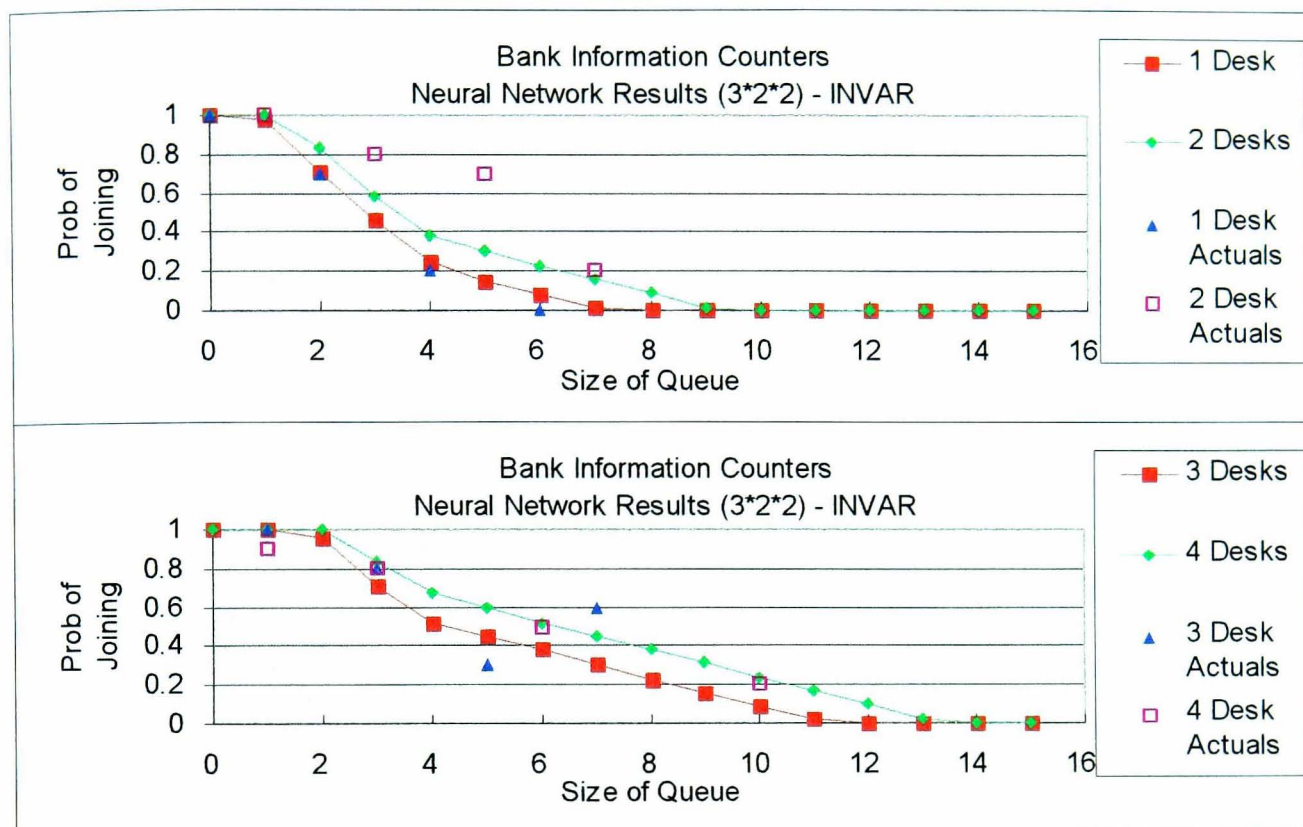
1	6	0.95	0	1	3	7	0.95	0	1
2	1	0.05	1	0	4	1	0.05	1	0
2	1	0.15	1	0	4	1	0.15	1	0
2	1	0.25	1	0	4	1	0.25	1	0
2	1	0.35	1	0	4	1	0.35	1	0
2	1	0.45	1	0	4	1	0.45	1	0
2	1	0.55	1	0	4	1	0.55	1	0
2	1	0.65	1	0	4	1	0.65	1	0
2	1	0.75	1	0	4	1	0.75	1	0
2	1	0.85	1	0	4	1	0.85	1	0
2	1	0.95	1	0	4	1	0.95	0	1
2	3	0.05	1	0	4	3	0.05	1	0
2	3	0.15	1	0	4	3	0.15	1	0
2	3	0.25	1	0	4	3	0.25	1	0
2	3	0.35	1	0	4	3	0.35	1	0
2	3	0.45	1	0	4	3	0.45	1	0
2	3	0.55	1	0	4	3	0.55	1	0
2	3	0.65	1	0	4	3	0.65	1	0
2	3	0.75	1	0	4	3	0.75	1	0
2	3	0.85	0	1	4	3	0.85	0	1
2	3	0.95	0	1	4	3	0.95	0	1
2	5	0.05	1	0	4	6	0.05	1	0
2	5	0.15	1	0	4	6	0.15	1	0
2	5	0.25	1	0	4	6	0.25	1	0
2	5	0.35	1	0	4	6	0.35	1	0
2	5	0.45	1	0	4	6	0.45	1	0
2	5	0.55	1	0	4	6	0.55	0	1
2	5	0.65	1	0	4	6	0.65	0	1
2	5	0.75	0	1	4	6	0.75	0	1
2	5	0.85	0	1	4	6	0.85	0	1
2	5	0.95	0	1	4	6	0.95	0	1
2	7	0.05	1	0	4	10	0.05	1	0
2	7	0.15	1	0	4	10	0.15	1	0
2	7	0.25	0	1	4	10	0.25	0	1
2	7	0.35	0	1	4	10	0.35	0	1
2	7	0.45	0	1	4	10	0.45	0	1
2	7	0.55	0	1	4	10	0.55	0	1
2	7	0.65	0	1	4	10	0.65	0	1
2	7	0.75	0	1	4	10	0.75	0	1
2	7	0.85	0	1	4	10	0.85	0	1
2	7	0.95	0	1	4	10	0.95	0	1

The results for the INVAR models, with different sizes of hidden layer are:

Counters Open	Queue Length	3*4*2		3*3*2		3*2*2		3*1*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	0	1	0	1	0	1	0	0.94	0.06	1	0
1	1	0.99	0.01	0.96	0.04	0.98	0.02	0.82	0.18		
1	2	0.7	0.3	0.7	0.3	0.71	0.29	0.69	0.31	0.7	0.3
1	3	0.41	0.59	0.45	0.55	0.46	0.54	0.57	0.43		
1	4	0.2	0.8	0.2	0.8	0.25	0.75	0.44	0.56	0.2	0.8
1	5	0.05	0.95	0	1	0.15	0.85	0.32	0.68		
1	6	0	1	0	1	0.08	0.92	0.19	0.81	0	1
1	7	0	1	0	1	0.01	0.99	0.07	0.93		
1	8	0	1	0	1	0	1	0	1		
1	9	0	1	0	1	0	1	0	1		
1	10	0	1	0	1	0	1	0	1		
1	11	0	1	0	1	0	1	0	1		
1	12	0	1	0	1	0	1	0	1		
1	13	0	1	0	1	0	1	0	1		
1	14	0	1	0.02	0.98	0	1	0	1		
1	15	0	1	0.04	0.96	0	1	0	1		
2	0	1	0	1	0	1	0	1	0		
2	1	1	0	1	0	1	0	0.95	0.05	1	0
2	2	0.92	0.08	1	0	0.83	0.17	0.82	0.18		
2	3	0.83	0.17	0.93	0.07	0.58	0.42	0.7	0.3	0.8	0.2
2	4	0.75	0.25	0.83	0.17	0.38	0.62	0.57	0.43		
2	5	0.67	0.33	0.68	0.32	0.3	0.7	0.45	0.55	0.7	0.3
2	6	0.53	0.47	0.45	0.55	0.23	0.77	0.32	0.68		
2	7	0.2	0.8	0.21	0.79	0.16	0.84	0.2	0.8	0.2	0.8
2	8	0	1	0.17	0.83	0.09	0.91	0.07	0.93		
2	9	0	1	0.24	0.76	0.01	0.99	0	1		
2	10	0	1	0.29	0.71	0	1	0	1		
2	11	0	1	0.26	0.74	0	1	0	1		
2	12	0	1	0.18	0.82	0	1	0	1		
2	13	0	1	0.09	0.91	0	1	0	1		
2	14	0	1	0.01	0.99	0	1	0	1		
2	15	0	1	0	1	0	1	0	1		
3	0	1	0	1	0	1	0	1	0		
3	1	0.96	0.04	0.97	0.03	1	0	1	0	1	0
3	2	0.88	0.12	0.89	0.11	0.95	0.05	0.95	0.05		
3	3	0.79	0.21	0.8	0.2	0.71	0.29	0.83	0.17	0.8	0.2
3	4	0.71	0.29	0.72	0.28	0.52	0.48	0.7	0.3		
3	5	0.63	0.37	0.64	0.36	0.45	0.55	0.58	0.42	0.3	0.7
3	6	0.55	0.45	0.6	0.4	0.38	0.62	0.45	0.55		
3	7	0.48	0.52	0.57	0.43	0.3	0.7	0.33	0.67	0.6	0.4
3	8	0.4	0.6	0.48	0.52	0.23	0.77	0.2	0.8		
3	9	0.32	0.68	0.38	0.62	0.16	0.84	0.08	0.92		
3	10	0.24	0.76	0.25	0.75	0.09	0.91	0	1		
3	11	0.12	0.88	0.14	0.86	0.02	0.98	0	1		
3	12	0	1	0.05	0.95	0	1	0	1		
3	13	0	1	0	1	0	1	0	1		
3	14	0	1	0	1	0	1	0	1		
3	15	0	1	0	1	0	1	0	1		
4	0	1	0	0.92	0.08	1	0	1	0		
4	1	0.93	0.07	0.86	0.14	1	0	1	0	0.9	0.1
4	2	0.84	0.16	0.84	0.16	1	0	1	0		
4	3	0.76	0.24	0.78	0.22	0.83	0.17	0.96	0.04	0.8	0.2
4	4	0.69	0.31	0.69	0.31	0.67	0.33	0.83	0.17		
4	5	0.61	0.39	0.61	0.39	0.59	0.41	0.71	0.29		
4	6	0.53	0.47	0.52	0.48	0.52	0.48	0.58	0.42	0.5	0.5
4	7	0.46	0.54	0.44	0.56	0.45	0.55	0.46	0.54		
4	8	0.38	0.62	0.35	0.65	0.38	0.62	0.33	0.67		
4	9	0.3	0.7	0.27	0.73	0.31	0.69	0.21	0.79		
4	10	0.22	0.78	0.18	0.82	0.24	0.76	0.08	0.92	0.2	0.8
4	11	0.14	0.86	0.1	0.9	0.17	0.83	0	1		
4	12	0.06	0.94	0.01	0.99	0.1	0.9	0	1		
4	13	0	1	0	1	0.02	0.98	0	1		
4	14	0	1	0	1	0	1	0	1		
4	15	0	1	0	1	0	1	0	1		

Table of Results for Information INVAR Networks

The model with the smoothest and most consistent fit is the $3*2*2$ model, which is shown graphically :



OUTDATA Models

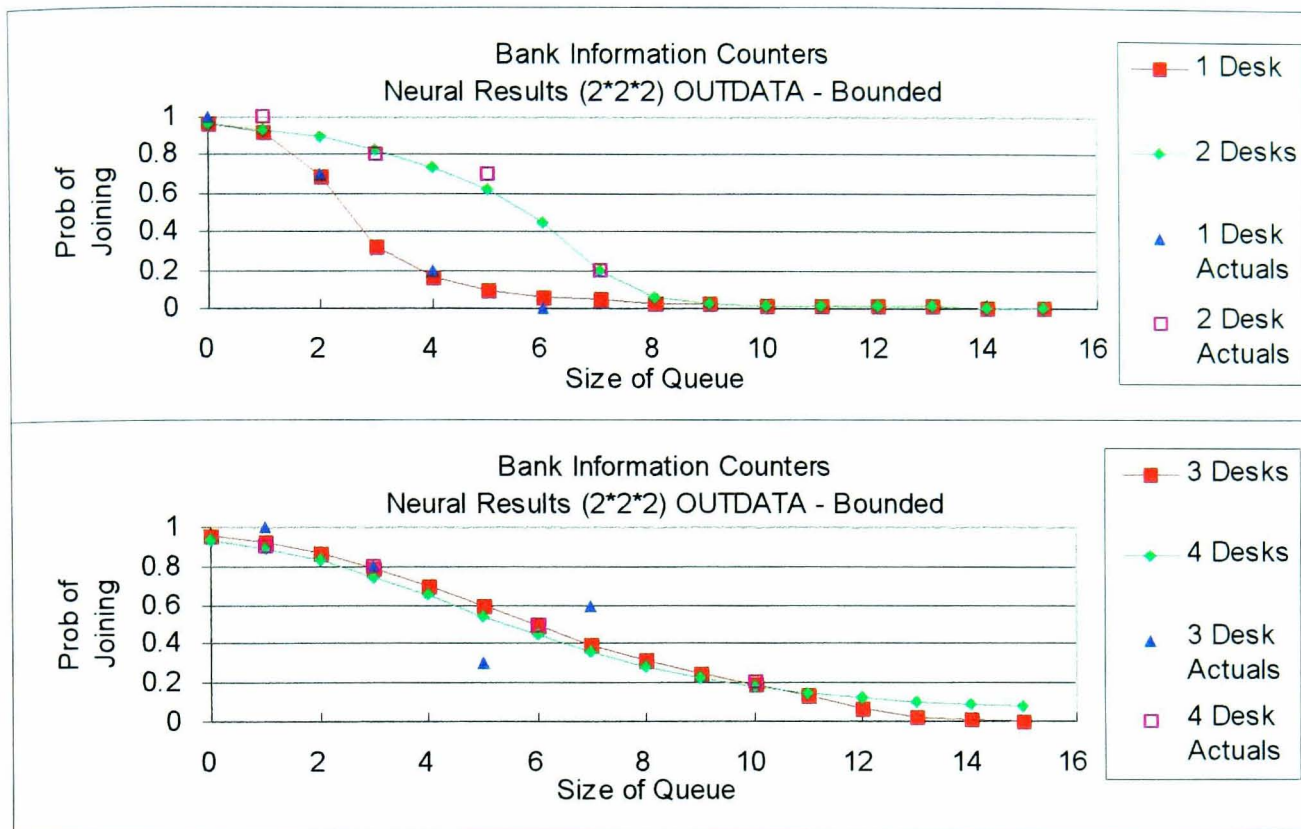
The unscaled training data for the OUTDATA model is:

Number of Counters	Single Queue Length	Probability of Staying	Probability of Leaving
1	0	1	0
1	2	0.7	0.3
1	4	0.2	0.8
1	6	0	1
2	1	1	0
2	3	0.8	0.2
2	5	0.7	0.3
2	7	0.2	0.8
3	1	1	0
3	3	0.8	0.2
3	5	0.3	0.7
3	7	0.6	0.4
4	1	0.9	0.1
4	3	0.8	0.2
4	6	0.5	0.5
4	10	0.2	0.8

Counters Open	Queue Length	2*4*2		2*3*2		2*2*2		2*1*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	0	0.99082	0.00918	0.97139	0.02861	0.99442	0.00558	0.92725	0.07275	1	0
1	1	0.92369	0.07631	0.95411	0.04589	0.96884	0.03116	0.83738	0.16262		
1	2	0.70432	0.29568	0.7037	0.2963	0.70358	0.29642	0.69997	0.30003	0.7	0.3
1	3	0.38846	0.61154	0.39598	0.60402	0.32072	0.67928	0.54433	0.45567		
1	4	0.20261	0.79739	0.17588	0.82412	0.17076	0.82924	0.40788	0.59212	0.2	0.8
1	5	0.03726	0.96274	0.08684	0.91316	0.10187	0.89813	0.30677	0.69323		
1	6	0.00212	0.99788	0.06308	0.93692	0.06652	0.93348	0.23784	0.76216	0	1
1	7	0.00017	0.99983	0.05673	0.94327	0.04724	0.95276	0.19212	0.80788		
1	8	0.00005	0.99995	0.05497	0.94503	0.03609	0.96391	0.16175	0.83825		
1	9	0.00003	0.99997	0.05447	0.94553	0.02929	0.97071	0.14129	0.85871		
1	10	0.00002	0.99998	0.05432	0.94568	0.02495	0.97505	0.12727	0.87273		
1	11	0.00002	0.99998	0.05428	0.94572	0.02207	0.97793	0.11751	0.88249		
1	12	0.00002	0.99998	0.05425	0.94575	0.02011	0.97989	0.11061	0.88939		
1	13	0.00002	0.99998	0.05419	0.94581	0.01874	0.98126	0.10568	0.89432		
1	14	0.00002	0.99998	0.05396	0.94604	0.01776	0.98224	0.10213	0.89787		
1	15	0.00002	0.99998	0.05317	0.94683	0.01706	0.98294	0.09954	0.90046		
2	0	0.99975	0.00025	0.97122	0.02878	0.99038	0.00962	0.97006	0.02994		
2	1	0.99966	0.00034	0.96644	0.03356	0.97025	0.02975	0.92297	0.07703	1	0
2	2	0.99775	0.00225	0.94788	0.05212	0.92469	0.07531	0.82978	0.17022		
2	3	0.78889	0.21111	0.87624	0.12376	0.84526	0.15474	0.69001	0.30999	0.8	0.2
2	4	0.36253	0.63747	0.71267	0.28733	0.73872	0.26128	0.53461	0.46539		
2	5	0.70903	0.29097	0.57524	0.42476	0.62217	0.37783	0.40024	0.59976	0.7	0.3
2	6	0.87633	0.12367	0.4938	0.5062	0.48204	0.51796	0.30142	0.69858		
2	7	0.20177	0.79823	0.20334	0.79666	0.2032	0.7968	0.23428	0.76572	0.2	0.8
2	8	0.00311	0.99689	0.06136	0.93864	0.04721	0.95279	0.18976	0.81024		
2	9	0.00018	0.99982	0.05465	0.94535	0.02814	0.97186	0.16017	0.83983		
2	10	0.00005	0.99995	0.05425	0.94575	0.02351	0.97649	0.14022	0.85978		
2	11	0.00003	0.99997	0.05404	0.94596	0.02103	0.97897	0.12653	0.87347		
2	12	0.00002	0.99998	0.05341	0.94659	0.01938	0.98062	0.11699	0.88301		
2	13	0.00002	0.99998	0.05124	0.94876	0.01822	0.98178	0.11024	0.88976		
2	14	0.00002	0.99998	0.0443	0.9557	0.01739	0.98261	0.10541	0.89459		
2	15	0.00002	0.99998	0.02761	0.97239	0.01679	0.98321	0.10193	0.89807		
3	0	0.99976	0.00024	0.96998	0.03002	0.98293	0.01707	0.98862	0.01138		
3	1	0.99969	0.00031	0.96176	0.03824	0.95201	0.04799	0.96806	0.03194	1	0
3	2	0.99768	0.00232	0.92906	0.07094	0.89025	0.10975	0.9185	0.0815		
3	3	0.82574	0.17426	0.81765	0.18235	0.7958	0.2042	0.82196	0.17804	0.8	0.2
3	4	0.47296	0.52704	0.64633	0.35367	0.6837	0.3163	0.67998	0.32002		
3	5	0.40402	0.59598	0.54653	0.45347	0.57546	0.42454	0.52499	0.47501	0.3	0.7
3	6	0.39082	0.60918	0.51147	0.48853	0.48481	0.51519	0.39275	0.60725		
3	7	0.59252	0.40748	0.50076	0.49924	0.41497	0.58503	0.29622	0.70378	0.6	0.4
3	8	0.94943	0.05057	0.49705	0.50295	0.36336	0.63664	0.23081	0.76919		
3	9	0.94718	0.05282	0.49379	0.50621	0.32578	0.67422	0.18747	0.81253		
3	10	0.56364	0.43636	0.48471	0.51529	0.29774	0.70226	0.15864	0.84136		
3	11	0.06315	0.93685	0.4483	0.5517	0.26953	0.73047	0.13917	0.86083		
3	12	0.00378	0.99622	0.25062	0.74938	0.1915	0.8085	0.1258	0.8742		
3	13	0.00037	0.99963	0.01755	0.98245	0.05156	0.94844	0.11648	0.88352		
3	14	0.00009	0.99991	0.00204	0.99796	0.02009	0.97991	0.10988	0.89012		
3	15	0.00004	0.99996	0.00049	0.99951	0.01685	0.98315	0.10515	0.89485		
4	0	0.99976	0.00024	0.96786	0.03214	0.97104	0.02896	0.9958	0.0042		
4	1	0.99967	0.00033	0.95354	0.04646	0.92629	0.07371	0.98781	0.01219	0.9	0.1
4	2	0.9968	0.0032	0.89692	0.10308	0.84776	0.15224	0.96594	0.03406		
4	3	0.78826	0.21174	0.74619	0.25381	0.74203	0.25797	0.91382	0.08618	0.8	0.2
4	4	0.47529	0.52471	0.59457	0.40543	0.62938	0.37062	0.81395	0.18605		
4	5	0.42793	0.57207	0.52665	0.47335	0.52872	0.47128	0.66989	0.33011		
4	6	0.40761	0.59239	0.50311	0.49689	0.44828	0.55172	0.51548	0.48452	0.5	0.5
4	7	0.36152	0.63848	0.49003	0.50997	0.38782	0.61218	0.38542	0.61458		
4	8	0.25728	0.74272	0.46449	0.53551	0.3436	0.6564	0.29114	0.70886		
4	9	0.13702	0.86298	0.38618	0.61382	0.31158	0.68842	0.22743	0.77257		
4	10	0.21129	0.78871	0.20342	0.79658	0.28838	0.71162	0.18523	0.81477	0.2	0.8
4	11	0.71333	0.28667	0.04618	0.95382	0.27151	0.72849	0.15714	0.84286		
4	12	0.78574	0.21426	0.01017	0.98983	0.25917	0.74083	0.13815	0.86185		
4	13	0.70208	0.29792	0.0048	0.9952	0.25011	0.74989	0.1251	0.8749		
4	14	0.56402	0.43598	0.00373	0.99627	0.2434	0.7566	0.11598	0.88402		
4	15	0.32589	0.67411	0.00346	0.99654	0.23825	0.76175	0.10952	0.89048		

Table of Results for Information OUTDATA Networks

The selected model has a 2*2*2 architecture, and is shown graphically :



Bounded OUTDATA Models

Artificial data was added to the training data set. These new data values were :

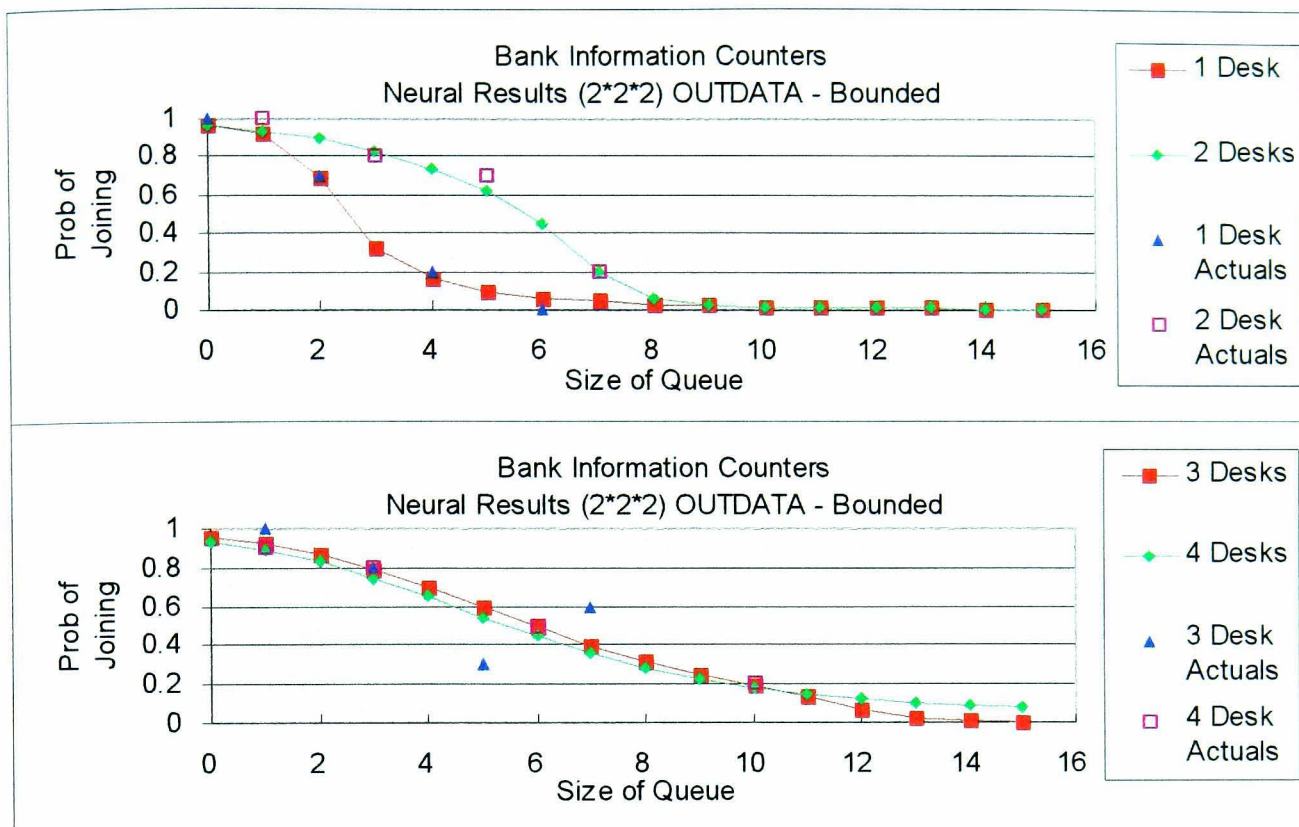
Number of Counters	Single Queue Length	Probability of Staying	Probability of Leaving
3	13	0	1
4	14	0	1

The results for the trained networks are shown overleaf.

Counters Open	Queue Length	2*4*2		2*2*2		2*1*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	0	0.99088	0.00912	0.96928	0.03072	0.89779	0.10221	1	0
1	1	0.95844	0.04156	0.91831	0.08169	0.8145	0.1855		
1	2	0.70092	0.29908	0.6953	0.3047	0.69432	0.30568	0.7	0.3
1	3	0.53042	0.46958	0.32387	0.67613	0.55046	0.44954		
1	4	0.2006	0.7994	0.15826	0.84174	0.40873	0.59127	0.2	0.8
1	5	0.00103	0.99897	0.09411	0.90589	0.2907	0.7093		
1	6	0.00002	0.99998	0.06074	0.93926	0.20346	0.79654	0	1
1	7	0	1	0.04077	0.95923	0.14329	0.85671		
1	8	0	1	0.02822	0.97178	0.10303	0.89697		
1	9	0	1	0.02012	0.97988	0.07622	0.92378		
1	10	0	1	0.01477	0.98523	0.05818	0.94182		
1	11	0	1	0.01116	0.98884	0.04584	0.95416		
1	12	0	1	0.00866	0.99134	0.03722	0.96278		
1	13	0	1	0.0069	0.9931	0.03106	0.96894		
1	14	0	1	0.00562	0.99438	0.02658	0.97342		
1	15	0	1	0.00468	0.99532	0.02326	0.97674		
2	0	0.99883	0.00117	0.96391	0.03609	0.94797	0.05203		
2	1	0.99813	0.00187	0.93599	0.06401	0.89872	0.10128	1	0
2	2	0.98986	0.01014	0.89192	0.10808	0.81596	0.18404		
2	3	0.80186	0.19814	0.82763	0.17237	0.69627	0.30373	0.8	0.2
2	4	0.27948	0.72052	0.74084	0.25916	0.55258	0.44742		
2	5	0.70183	0.29817	0.62592	0.37408	0.41064	0.58936	0.7	0.3
2	6	0.80246	0.19754	0.45121	0.54879	0.29217	0.70783		
2	7	0.19644	0.80356	0.19546	0.80454	0.2045	0.7955	0.2	0.8
2	8	0.03298	0.96702	0.05211	0.94789	0.144	0.856		
2	9	0.01248	0.98752	0.02138	0.97862	0.1035	0.8965		
2	10	0.00753	0.99247	0.01348	0.98652	0.07653	0.92347		
2	11	0.00529	0.99471	0.00993	0.99007	0.0584	0.9416		
2	12	0.00364	0.99636	0.00774	0.99226	0.04599	0.95401		
2	13	0.00219	0.99781	0.00622	0.99378	0.03732	0.96268		
2	14	0.00107	0.99893	0.00513	0.99487	0.03114	0.96886		
2	15	0.00041	0.99959	0.00431	0.99569	0.02664	0.97336		
3	0	0.99825	0.00175	0.95193	0.04807	0.97461	0.02539		
3	1	0.99058	0.00942	0.9167	0.0833	0.94848	0.05152	1	0
3	2	0.91753	0.08247	0.86329	0.13671	0.89965	0.10035		
3	3	0.78879	0.21121	0.78943	0.21057	0.81743	0.18257	0.8	0.2
3	4	0.62904	0.37096	0.69746	0.30254	0.69821	0.30179		
3	5	0.30978	0.69022	0.59499	0.40501	0.5547	0.4453	0.3	0.7
3	6	0.1081	0.8919	0.49236	0.50764	0.41255	0.58745		
3	7	0.60171	0.39829	0.39858	0.60142	0.29366	0.70634	0.6	0.4
3	8	0.48494	0.51506	0.3186	0.6814	0.20556	0.79444		
3	9	0.08204	0.91796	0.25269	0.74731	0.14471	0.85529		
3	10	0.02198	0.97802	0.1961	0.8039	0.10398	0.89602		
3	11	0.01195	0.98805	0.13618	0.86382	0.07685	0.92315		
3	12	0.00927	0.99073	0.06325	0.93675	0.05861	0.94139		
3	13	0.00838	0.99162	0.01743	0.98257	0.04614	0.95386	0	1
3	14	0.00804	0.99196	0.00662	0.99338	0.03742	0.96258		
3	15	0.00791	0.99209	0.00433	0.99567	0.03121	0.96879		
4	0	0.98393	0.01607	0.93665	0.06335	0.98784	0.01216		
4	1	0.90352	0.09648	0.89299	0.10701	0.97487	0.02513	0.9	0.1
4	2	0.83655	0.16345	0.82958	0.17042	0.94899	0.05101		
4	3	0.81147	0.18853	0.74619	0.25381	0.90056	0.09944	0.8	0.2
4	4	0.78277	0.21723	0.64786	0.35214	0.81888	0.18112		
4	5	0.70806	0.29194	0.54402	0.45598	0.70015	0.29985		
4	6	0.49208	0.50792	0.44488	0.55512	0.55682	0.44318	0.5	0.5
4	7	0.14039	0.85961	0.35775	0.64225	0.41446	0.58554		
4	8	0.04865	0.95135	0.28575	0.71425	0.29515	0.70485		
4	9	0.43412	0.56588	0.22865	0.77135	0.20662	0.79338		
4	10	0.20428	0.79572	0.18444	0.81556	0.14543	0.85457	0.2	0.8
4	11	0.03936	0.96064	0.15059	0.84941	0.10445	0.89555		
4	12	0.01549	0.98451	0.12473	0.87527	0.07717	0.92283		
4	13	0.01032	0.98968	0.10484	0.89516	0.05883	0.94117		
4	14	0.00875	0.99125	0.08915	0.91085	0.04628	0.95372	0	1
4	15	0.00819	0.99181	0.07554	0.92446	0.03753	0.96247		

Table of Results for Bounded Information OUTDATA Networks

The selected bounded network is the 2*2*2 network, which is shown graphically :



OUTNET Models

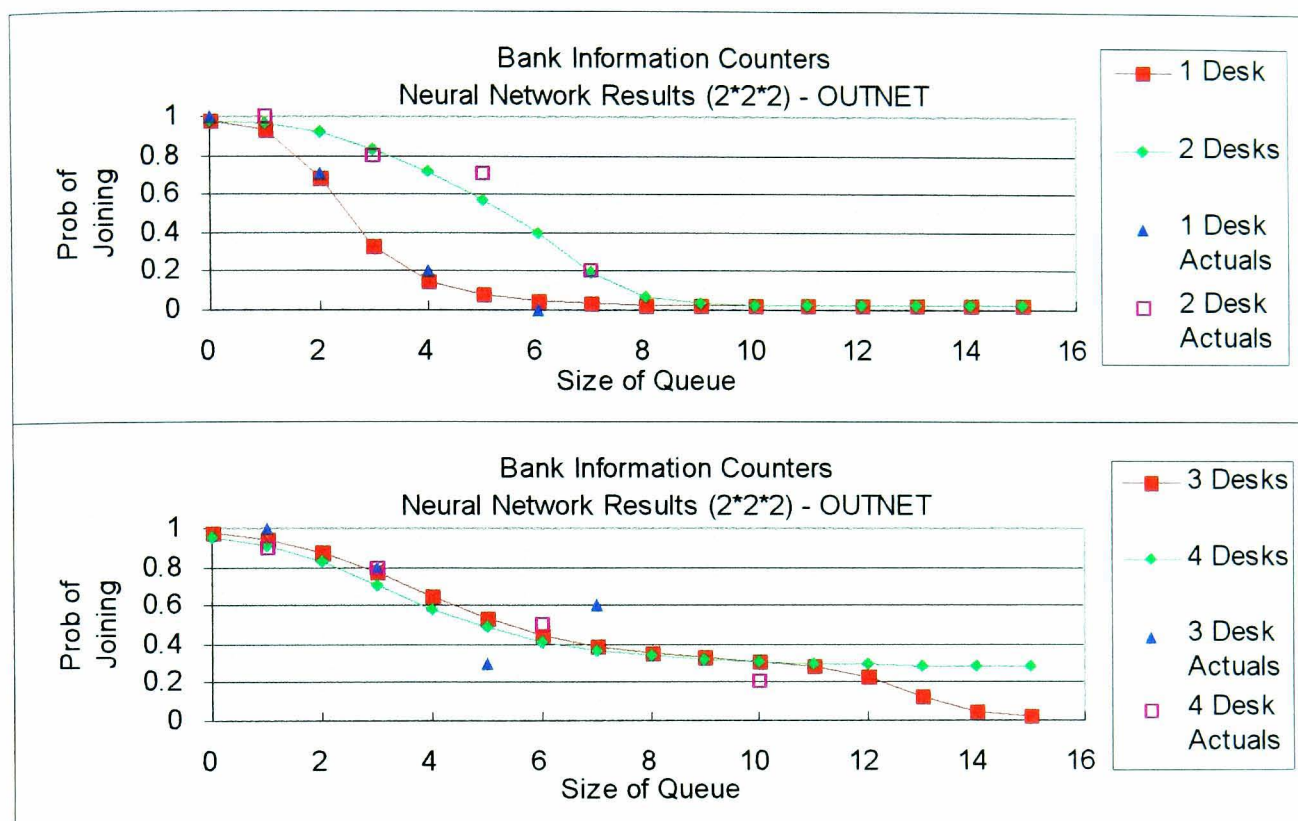
The training data for the OUTNET models is the same as for the INVAR models, except that the probability input variable is removed (see page H-1).

The results of training neural networks with a variety of numbers of nodes in the hidden layer are shown overleaf.

Counters	Queue	2*4*2		2*3*2		2*2*2		2*1*2		Training	
		Open	Length	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	0			0.96769	0.03231	0.95515	0.04485	0.98222	0.01778	0.92001	0.07999
1	1			0.93394	0.06606	0.82983	0.17017	0.93104	0.06896	0.84099	0.15901
1	2			0.71076	0.28924	0.70767	0.29233	0.68462	0.31538	0.70331	0.29669
1	3			0.2798	0.7202	0.56533	0.43467	0.32699	0.67301	0.53442	0.46558
1	4			0.14221	0.85779	0.19312	0.80688	0.15158	0.84842	0.39119	0.60881
1	5			0.10556	0.89444	0.02685	0.97315	0.08144	0.91856	0.2967	0.7033
1	6			0.07259	0.92741	0.01098	0.98902	0.05067	0.94933	0.24107	0.75893
1	7			0.03167	0.96833	0.00896	0.99104	0.03583	0.96417	0.20932	0.79067
1	8			0.00882	0.99118	0.00862	0.99138	0.02812	0.97188	0.19119	0.80881
1	9			0.00315	0.99685	0.00856	0.99144	0.02385	0.97615	0.18074	0.81926
1	10			0.00195	0.99805	0.00855	0.99145	0.02138	0.97862	0.17467	0.82533
1	11			0.00165	0.99835	0.00854	0.99146	0.01991	0.98009	0.17112	0.82888
1	12			0.00156	0.99844	0.00854	0.99146	0.01901	0.98099	0.16903	0.83097
1	13			0.00154	0.99846	0.00854	0.99146	0.01846	0.98154	0.1678	0.8322
1	14			0.00153	0.99847	0.00854	0.99146	0.01811	0.98189	0.16708	0.83292
1	15			0.00153	0.99847	0.00854	0.99146	0.01789	0.98211	0.16665	0.83335
2	0			0.97243	0.02757	0.98108	0.01892	0.982	0.018	0.95488	0.04512
2	1			0.96463	0.03537	0.98051	0.01949	0.96197	0.03803	0.91522	0.08478
2	2			0.92108	0.07892	0.94117	0.05883	0.91761	0.08239	0.83144	0.16856
2	3			0.79518	0.20482	0.80621	0.19379	0.83423	0.16577	0.68916	0.31084
2	4			0.76695	0.23305	0.7202	0.2798	0.71213	0.28787	0.52039	0.47961
2	5			0.70396	0.29604	0.68084	0.31916	0.56874	0.43126	0.38113	0.61887
2	6			0.45154	0.54846	0.56235	0.43765	0.40124	0.59876	0.29061	0.70939
2	7			0.17812	0.82188	0.19622	0.80378	0.19821	0.80179	0.23757	0.76243
2	8			0.04879	0.95121	0.02726	0.97274	0.06744	0.93256	0.20733	0.79267
2	9			0.01066	0.98934	0.01102	0.98898	0.03114	0.96886	0.19005	0.80995
2	10			0.0034	0.9966	0.00896	0.99104	0.02247	0.97753	0.18008	0.81992
2	11			0.002	0.998	0.00862	0.99138	0.01984	0.98016	0.17428	0.82572
2	12			0.00166	0.99834	0.00856	0.99144	0.0188	0.9812	0.17089	0.82911
2	13			0.00157	0.99843	0.00855	0.99145	0.01828	0.98172	0.1689	0.8311
2	14			0.00154	0.99846	0.00854	0.99146	0.01799	0.98201	0.16773	0.83227
2	15			0.00153	0.99847	0.00854	0.99146	0.01781	0.98219	0.16703	0.83297
3	0			0.97306	0.02694	0.89189	0.10811	0.97335	0.02665	0.97154	0.02846
3	1			0.96753	0.03247	0.8905	0.1095	0.94231	0.05769	0.95258	0.04742
3	2			0.93357	0.06643	0.88178	0.11822	0.87868	0.12132	0.9101	0.0899
3	3			0.74975	0.25025	0.82578	0.17422	0.77496	0.22504	0.82142	0.17858
3	4			0.49597	0.50403	0.55637	0.44363	0.64938	0.35062	0.67479	0.32521
3	5			0.41865	0.58135	0.29268	0.70732	0.53425	0.46575	0.50661	0.49339
3	6			0.44173	0.55827	0.3385	0.6615	0.44735	0.55265	0.37146	0.62854
3	7			0.57761	0.42239	0.59191	0.40809	0.38833	0.61167	0.28479	0.71521
3	8			0.72541	0.27459	0.66334	0.33666	0.34993	0.65007	0.23425	0.76575
3	9			0.63219	0.36781	0.56218	0.43782	0.32462	0.67538	0.20544	0.79456
3	10			0.25664	0.74336	0.19972	0.80028	0.30517	0.69483	0.18896	0.81104
3	11			0.04738	0.95262	0.02768	0.97232	0.28017	0.71983	0.17945	0.82055
3	12			0.00963	0.99037	0.01107	0.98893	0.22489	0.77511	0.17392	0.82608
3	13			0.00343	0.99657	0.00897	0.99103	0.12309	0.87691	0.17068	0.82932
3	14			0.00208	0.99792	0.00862	0.99138	0.04736	0.95264	0.16877	0.83123
3	15			0.00171	0.99829	0.00856	0.99144	0.02464	0.97536	0.16765	0.83235
4	0			0.97349	0.02651	0.89215	0.10785	0.95979	0.04021	0.97974	0.02026
4	1			0.96987	0.03013	0.8921	0.1079	0.91317	0.08683	0.97043	0.02957
4	2			0.94858	0.05142	0.89179	0.10821	0.82757	0.17243	0.95011	0.04989
4	3			0.81766	0.18234	0.88984	0.11016	0.70821	0.29179	0.90463	0.09537
4	4			0.54047	0.45953	0.8776	0.1224	0.58469	0.41531	0.81093	0.18907
4	5			0.41867	0.58133	0.799	0.201	0.48389	0.51611	0.66025	0.33975
4	6			0.38934	0.61066	0.48246	0.51754	0.41271	0.58729	0.49311	0.50689
4	7			0.37667	0.62333	0.25118	0.74882	0.36589	0.63411	0.36216	0.63784
4	8			0.35334	0.64666	0.20514	0.79486	0.33596	0.66404	0.27924	0.72076
4	9			0.29416	0.70584	0.19721	0.80279	0.31702	0.68298	0.23107	0.76893
4	10			0.19274	0.80726	0.19337	0.80663	0.30504	0.69496	0.20363	0.79637
4	11			0.12585	0.87415	0.1797	0.8203	0.29746	0.70254	0.18792	0.81208
4	12			0.12167	0.87833	0.12644	0.87356	0.29265	0.70735	0.17885	0.82115
4	13			0.11782	0.88218	0.04223	0.95777	0.28954	0.71046	0.17356	0.82644
4	14			0.10239	0.89761	0.01522	0.98478	0.28735	0.71265	0.17047	0.82953
4	15			0.07625	0.92375	0.01014	0.98986	0.28509	0.71491	0.16865	0.83135

Table of Results for Information OUTNET Networks

The chosen OUTNET network has a 2*2*2 architecture, and is shown graphically :



Bounded OUTNET Models

Artificial data was added to the OUTNET training data set, as shown below.

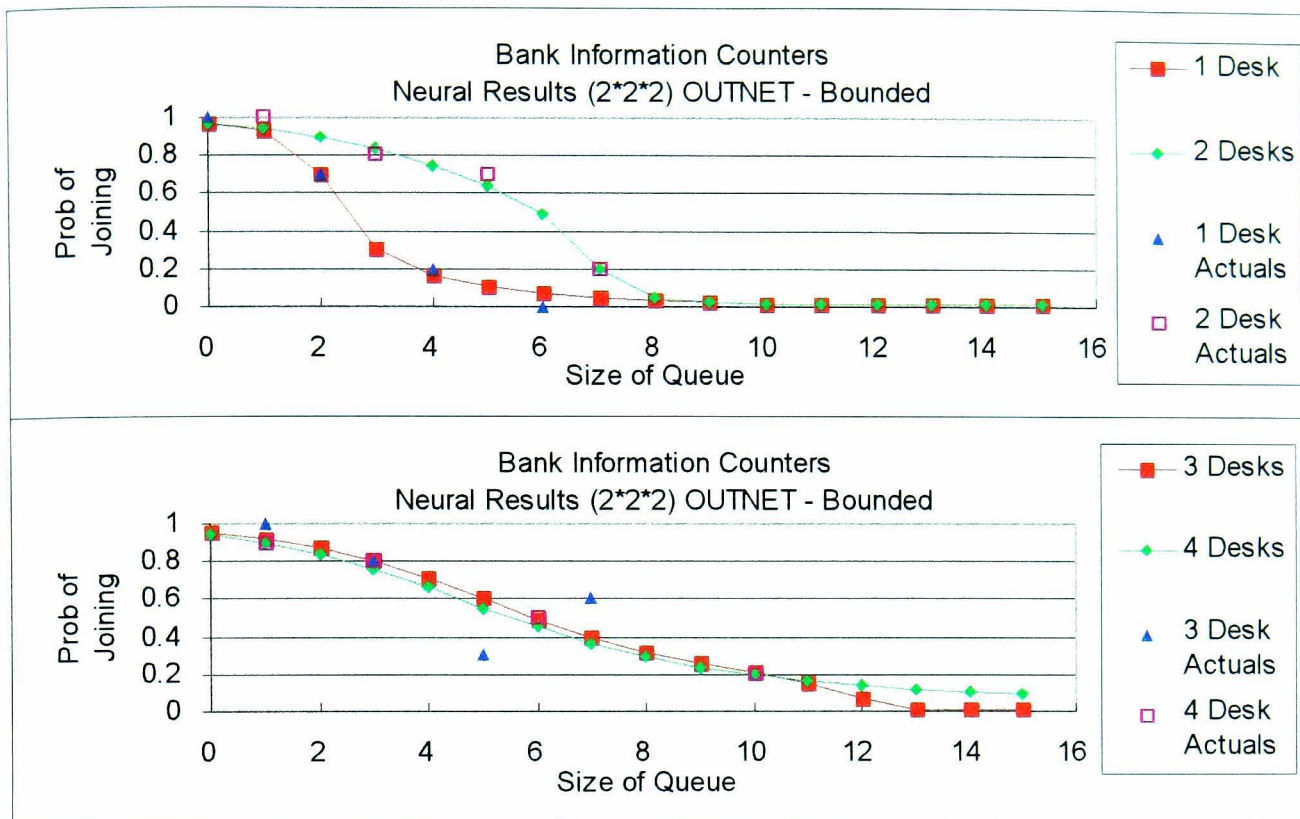
[illegible]

The results for the bounded OUTNET models are:

Counters	Queue Open	Queue Length	2*4*2		2*2*2		2*1*2		Training	
			Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	0	0.98181	0.01819	0.96948	0.03052	0.89031	0.10969	1	0
1	1	1	0.91985	0.08015	0.9301	0.0699	0.81052	0.18948		
1	1	2	0.7206	0.2794	0.69674	0.30326	0.69155	0.30845	0.7	0.3
1	1	3	0.30681	0.69319	0.30466	0.69534	0.54503	0.45497		
1	1	4	0.18252	0.81748	0.1663	0.8337	0.40009	0.59991	0.2	0.8
1	1	5	0.14757	0.85243	0.10471	0.89529	0.28224	0.71776		
1	1	6	0.00907	0.99093	0.0683	0.9317	0.1986	0.8014	0	1
1	1	7	0.00001	0.99999	0.04579	0.95421	0.14339	0.85661		
1	1	8	0	1	0.03171	0.96829	0.10784	0.89216		
1	1	9	0	1	0.02278	0.97722	0.08487	0.91513		
1	1	10	0	1	0.01698	0.98302	0.06978	0.93022		
1	1	11	0	1	0.01313	0.98687	0.05965	0.94035		
1	1	12	0	1	0.0105	0.9895	0.0527	0.9473		
1	1	13	0	1	0.00867	0.99133	0.04784	0.95216		
1	1	14	0	1	0.00736	0.99264	0.04438	0.95562		
1	1	15	0	1	0.0064	0.9936	0.04189	0.95811		
2	2	0	0.96221	0.03779	0.96306	0.03694	0.93748	0.06252		
2	2	1	0.95823	0.04177	0.93693	0.06307	0.89059	0.10941	1	0
2	2	2	0.92453	0.07547	0.89493	0.10507	0.81097	0.18903		
2	2	3	0.80693	0.19307	0.83225	0.16775	0.69217	0.30783	0.8	0.2
2	2	4	0.73844	0.26156	0.74714	0.25286	0.54571	0.45429		
2	2	5	0.70569	0.29431	0.64001	0.35999	0.4007	0.5993	0.7	0.3
2	2	6	0.65306	0.34694	0.48576	0.51424	0.28269	0.71731		
2	2	7	0.20468	0.79532	0.19922	0.80078	0.19891	0.80109	0.2	0.8
2	2	8	0.0001	0.9999	0.04378	0.95622	0.14359	0.85641		
2	2	9	0	1	0.02136	0.97864	0.10797	0.89203		
2	2	10	0	1	0.01523	0.98477	0.08495	0.91505		
2	2	11	0	1	0.01186	0.98814	0.06983	0.93017		
2	2	12	0	1	0.00961	0.99039	0.05968	0.94032		
2	2	13	0	1	0.00804	0.99196	0.05272	0.94728		
2	2	14	0	1	0.0069	0.9931	0.04786	0.95214		
2	2	15	0	1	0.00606	0.99394	0.0444	0.9556		
3	3	0	0.96243	0.03757	0.95314	0.04686	0.96346	0.03654		
3	3	1	0.95997	0.04003	0.92068	0.07932	0.93764	0.06236	1	0
3	3	2	0.93866	0.06134	0.87005	0.12995	0.89087	0.10913		
3	3	3	0.76411	0.23589	0.79759	0.20241	0.81142	0.18858	0.8	0.2
3	3	4	0.48546	0.51454	0.70467	0.29533	0.69278	0.30722		
3	3	5	0.41957	0.58043	0.59933	0.40067	0.54639	0.45361	0.3	0.7
3	3	6	0.42291	0.57709	0.49366	0.50634	0.40131	0.59869		
3	3	7	0.60595	0.39405	0.3984	0.6016	0.28315	0.71685	0.6	0.4
3	3	8	0.65864	0.34136	0.31921	0.68079	0.19922	0.80078		
3	3	9	0.05721	0.94279	0.25665	0.74335	0.14379	0.85621		
3	3	10	0.00008	0.99992	0.20694	0.79306	0.1081	0.8919		
3	3	11	0.00004	0.99996	0.15659	0.84341	0.08504	0.91496		
3	3	12	0.00003	0.99997	0.0733	0.9267	0.06989	0.93011		
3	3	13	0.00002	0.99998	0.01594	0.98406	0.05972	0.94028	0	1
3	3	14	0	1	0.00734	0.99266	0.05275	0.94725		
3	3	15	0	1	0.00586	0.99414	0.04787	0.95213		
4	4	0	0.96256	0.03744	0.94072	0.05928	0.97754	0.02246		
4	4	1	0.96105	0.03895	0.90087	0.09913	0.96354	0.03646	0.9	0.1
4	4	2	0.94832	0.05168	0.84084	0.15916	0.9378	0.0622		
4	4	3	0.83327	0.16673	0.75879	0.24121	0.89115	0.10885	0.8	0.2
4	4	4	0.52704	0.47296	0.65904	0.34096	0.81187	0.18813		
4	4	5	0.4255	0.5745	0.55202	0.44798	0.6934	0.3066		
4	4	6	0.41112	0.58888	0.44988	0.55012	0.54708	0.45292	0.5	0.5
4	4	7	0.40928	0.59072	0.36136	0.63864	0.40192	0.59808		
4	4	8	0.4087	0.5913	0.28981	0.71019	0.28361	0.71639		
4	4	9	0.39936	0.60064	0.23447	0.76553	0.19953	0.80047		
4	4	10	0.20578	0.79422	0.19268	0.80732	0.14399	0.85601	0.2	0.8
4	4	11	0.00044	0.99956	0.1614	0.8386	0.10823	0.89177		
4	4	12	0.00004	0.99996	0.138	0.862	0.08512	0.91488		
4	4	13	0.00004	0.99996	0.12038	0.87962	0.06994	0.93006		
4	4	14	0.00004	0.99996	0.10692	0.89308	0.05976	0.94024	0	1
4	4	15	0.00004	0.99996	0.09599	0.90401	0.05277	0.94723		

Table of Results for Bounded Information OUTNET Networks

The chosen model has a 2*2*2 architecture :



H2 : Transactions Service

The training data and models are presented for the Stay/Balk decisions for the Information service. The INVAR, OUTDATA and OUTNET models will be dealt with separately.

INVAR Models

The unscaled training data for the INVAR models is shown below.

Number Counters	Shortest Length	Probability	Stay Decision	Balk Decision	Number Counters	Shortest Queue	Probability	Stay Decision	Balk Decision
1	1	0.05	1	0	3	2	0.825	1	0
1	1	0.15	1	0	3	2	0.875	1	0
1	1	0.25	1	0	3	2	0.925	1	0
1	1	0.35	1	0	3	2	0.975	1	0
1	1	0.45	1	0	3	4	0.05	1	0
1	1	0.55	1	0	3	4	0.15	1	0
1	1	0.65	1	0	3	4	0.25	1	0
1	1	0.75	1	0	3	4	0.35	1	0
1	1	0.85	1	0	3	4	0.45	1	0
1	1	0.95	1	0	3	4	0.55	1	0
1	3	0.05	1	0	3	4	0.65	0	1
1	3	0.15	1	0	3	4	0.75	0	1
1	3	0.25	1	0	3	4	0.85	0	1
1	3	0.35	1	0	3	4	0.95	0	1
1	3	0.45	1	0	3	7	0.05	1	0
1	3	0.55	1	0	3	7	0.15	1	0
1	3	0.65	1	0	3	7	0.25	1	0
1	3	0.75	0	1	3	7	0.35	0	1
1	3	0.85	0	1	3	7	0.45	0	1
1	3	0.95	0	1	3	7	0.55	0	1
1	5	0.05	1	0	3	7	0.65	0	1
1	5	0.15	1	0	3	7	0.75	0	1
1	5	0.25	1	0	3	7	0.85	0	1
1	5	0.35	1	0	3	7	0.95	0	1
1	5	0.45	1	0	4	2	0.05	1	0
1	5	0.55	0	1	4	2	0.15	1	0
1	5	0.65	0	1	4	2	0.25	1	0
1	5	0.75	0	1	4	2	0.35	1	0
1	5	0.85	0	1	4	2	0.45	1	0
1	5	0.95	0	1	4	2	0.55	1	0
1	7	0.05	1	0	4	2	0.65	1	0
1	7	0.15	1	0	4	2	0.75	1	0
1	7	0.25	1	0	4	2	0.85	1	0
1	7	0.35	1	0	4	2	0.95	1	0
1	7	0.45	0	1	4	3	0.05	1	0
1	7	0.55	0	1	4	3	0.15	1	0
1	7	0.65	0	1	4	3	0.25	1	0
1	7	0.75	0	1	4	3	0.35	1	0
1	7	0.85	0	1	4	3	0.45	1	0
1	7	0.95	0	1	4	3	0.55	1	0
1	9	0.05	1	0	4	3	0.65	1	0
1	9	0.15	1	0	4	3	0.75	0	1
1	9	0.25	1	0	4	3	0.85	0	1
1	9	0.35	1	0	4	3	0.95	0	1
1	9	0.45	0	1	4	5	0.05	1	0
1	9	0.55	0	1	4	5	0.15	1	0
1	9	0.65	0	1	4	5	0.25	1	0
1	9	0.75	0	1	4	5	0.35	1	0
1	9	0.85	0	1	4	5	0.45	1	0
1	9	0.95	0	1	4	5	0.55	1	0
2	2	0.05	1	0	4	5	0.65	0	1
2	2	0.15	1	0	4	5	0.75	0	1
2	2	0.25	1	0	4	5	0.85	0	1

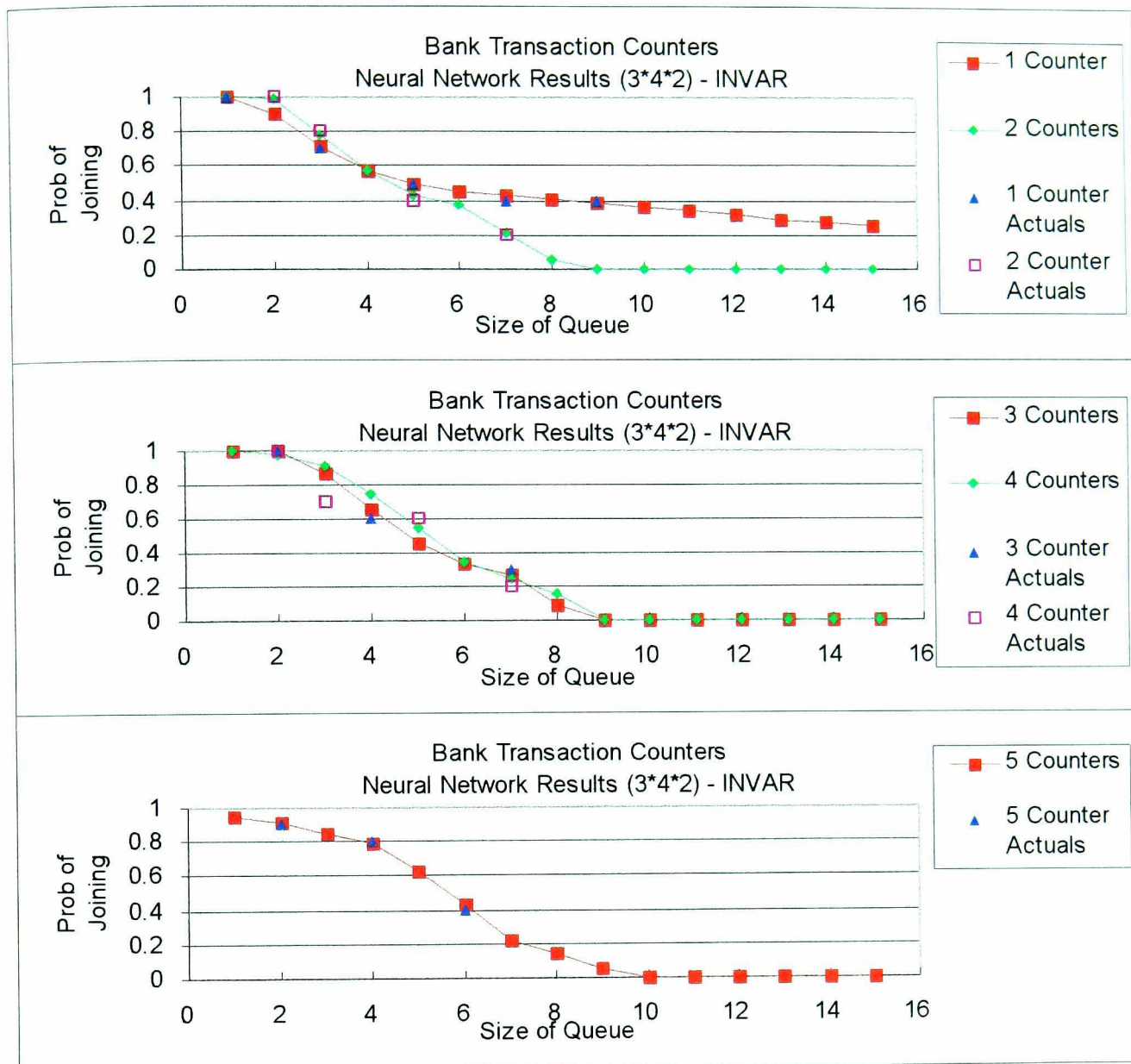
2	2	0.35	1	0	4	5	0.95	0	1
2	2	0.45	1	0	4	7	0.05	1	0
2	2	0.55	1	0	4	7	0.15	1	0
2	2	0.65	1	0	4	7	0.25	0	1
2	2	0.75	1	0	4	7	0.35	0	1
2	2	0.85	1	0	4	7	0.45	0	1
2	2	0.95	1	0	4	7	0.55	0	1
2	3	0.05	1	0	4	7	0.65	0	1
2	3	0.15	1	0	4	7	0.75	0	1
2	3	0.25	1	0	4	7	0.85	0	1
2	3	0.35	1	0	4	7	0.95	0	1
2	3	0.45	1	0	5	2	0.025	1	0
2	3	0.55	1	0	5	2	0.075	1	0
2	3	0.65	1	0	5	2	0.125	1	0
2	3	0.75	1	0	5	2	0.175	1	0
2	3	0.85	0	1	5	2	0.225	1	0
2	3	0.95	0	1	5	2	0.275	1	0
2	5	0.05	1	0	5	2	0.325	1	0
2	5	0.15	1	0	5	2	0.375	1	0
2	5	0.25	1	0	5	2	0.425	1	0
2	5	0.35	1	0	5	2	0.475	1	0
2	5	0.45	0	1	5	2	0.525	1	0
2	5	0.55	0	1	5	2	0.575	1	0
2	5	0.65	0	1	5	2	0.625	1	0
2	5	0.75	0	1	5	2	0.675	1	0
2	5	0.85	0	1	5	2	0.725	1	0
2	5	0.95	0	1	5	2	0.775	1	0
2	7	0.05	1	0	5	2	0.825	1	0
2	7	0.15	1	0	5	2	0.875	1	0
2	7	0.25	0	1	5	2	0.925	0	1
2	7	0.35	0	1	5	2	0.975	0	1
2	7	0.45	0	1	5	4	0.05	1	0
2	7	0.55	0	1	5	4	0.15	1	0
2	7	0.65	0	1	5	4	0.25	1	0
2	7	0.75	0	1	5	4	0.35	1	0
2	7	0.85	0	1	5	4	0.45	1	0
2	7	0.95	0	1	5	4	0.55	1	0
3	2	0.025	1	0	5	4	0.65	1	0
3	2	0.075	1	0	5	4	0.75	1	0
3	2	0.125	1	0	5	4	0.85	0	1
3	2	0.175	1	0	5	4	0.95	0	1
3	2	0.225	1	0	5	6	0.05	1	0
3	2	0.275	1	0	5	6	0.15	1	0
3	2	0.325	1	0	5	6	0.25	1	0
3	2	0.375	1	0	5	6	0.35	1	0
3	2	0.425	1	0	5	6	0.45	0	1
3	2	0.475	1	0	5	6	0.55	0	1
3	2	0.525	1	0	5	6	0.65	0	1
3	2	0.575	1	0	5	6	0.75	0	1
3	2	0.625	1	0	5	6	0.85	0	1
3	2	0.675	1	0	5	6	0.95	0	1
3	2	0.725	1	0	5	6	0.95	0	1
3	2	0.775	1	0	5	6	0.95	0	1

The results for the INVAR models, with different sizes of hidden layer are:

Counters Open	Queue Length	3*6*2		3*4*2		3*3*2		3*2*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	1	0	1	0	1	0	1	0	1	0
1	2	0.97	0.03	0.9	0.1	0.87	0.13	0.91	0.09		
1	3	0.72	0.28	0.71	0.29	0.67	0.33	0.73	0.27	0.7	0.3
1	4	0.53	0.47	0.57	0.43	0.55	0.45	0.55	0.45		
1	5	0.48	0.52	0.49	0.51	0.47	0.53	0.46	0.54	0.5	0.5
1	6	0.48	0.52	0.45	0.55	0.43	0.57	0.42	0.58		
1	7	0.4	0.6	0.43	0.57	0.41	0.59	0.41	0.59	0.4	0.6
1	8	0.39	0.61	0.41	0.59	0.39	0.61	0.39	0.61		
1	9	0.39	0.61	0.39	0.61	0.37	0.63	0.38	0.62	0.4	0.6
1	10	0.38	0.62	0.36	0.64	0.36	0.64	0.36	0.64		
1	11	0.38	0.62	0.34	0.66	0.34	0.66	0.34	0.66		
1	12	0.37	0.63	0.32	0.68	0.31	0.69	0.33	0.67		
1	13	0.36	0.64	0.29	0.71	0.27	0.73	0.31	0.69		
1	14	0.36	0.64	0.27	0.73	0.2	0.8	0.3	0.7		
1	15	0.35	0.65	0.25	0.75	0.12	0.88	0.28	0.72		
2	1	1	0	1	0	1	0	1	0		
2	2	1	0	0.99	0.01	0.98	0.02	0.98	0.02	1	0
2	3	0.81	0.19	0.78	0.22	0.76	0.24	0.79	0.21	0.8	0.2
2	4	0.57	0.43	0.57	0.43	0.55	0.45	0.61	0.39		
2	5	0.39	0.61	0.43	0.57	0.42	0.58	0.44	0.56	0.4	0.6
2	6	0.23	0.77	0.37	0.63	0.36	0.64	0.36	0.64		
2	7	0.22	0.78	0.21	0.79	0.34	0.66	0.34	0.66	0.2	0.8
2	8	0.21	0.79	0.05	0.95	0.32	0.68	0.32	0.68		
2	9	0.21	0.79	0	1	0.3	0.7	0.31	0.69		
2	10	0.2	0.8	0	1	0.28	0.72	0.29	0.71		
2	11	0.21	0.79	0	1	0.27	0.73	0.27	0.73		
2	12	0.21	0.79	0	1	0.24	0.76	0.26	0.74		
2	13	0.22	0.78	0	1	0.21	0.79	0.24	0.76		
2	14	0.24	0.76	0	1	0.14	0.86	0.23	0.77		
2	15	0.25	0.75	0	1	0.06	0.94	0.21	0.79		
3	1	1	0	1	0	1	0	1	0		
3	2	1	0	1	0	1	0	1	0	1	0
3	3	0.9	0.1	0.87	0.13	0.86	0.14	0.86	0.14		
3	4	0.59	0.41	0.66	0.34	0.65	0.35	0.67	0.33	0.6	0.4
3	5	0.44	0.56	0.46	0.54	0.43	0.57	0.49	0.51		
3	6	0.31	0.69	0.33	0.67	0.31	0.69	0.34	0.66		
3	7	0.31	0.69	0.27	0.73	0.26	0.74	0.27	0.73	0.3	0.7
3	8	0.34	0.66	0.09	0.91	0.24	0.76	0.25	0.75		
3	9	0.35	0.65	0	1	0.23	0.77	0.24	0.76		
3	10	0.36	0.64	0	1	0.21	0.79	0.22	0.78		
3	11	0.36	0.64	0	1	0.19	0.81	0.21	0.79		
3	12	0.35	0.65	0	1	0.17	0.83	0.19	0.81		
3	13	0.34	0.66	0	1	0.14	0.86	0.17	0.83		
3	14	0.33	0.67	0	1	0.08	0.92	0.16	0.84		
3	15	0.31	0.69	0	1	0.01	0.99	0.14	0.86		
4	1	1	0	1	0	1	0	1	0		
4	2	1	0	0.98	0.02	0.97	0.03	1	0	1	0
4	3	0.72	0.28	0.91	0.09	0.87	0.13	0.93	0.07	0.7	0.3
4	4	0.69	0.31	0.75	0.25	0.74	0.26	0.74	0.26		
4	5	0.57	0.43	0.54	0.46	0.54	0.46	0.56	0.44	0.6	0.4
4	6	0.33	0.67	0.34	0.66	0.32	0.68	0.37	0.63		
4	7	0.19	0.81	0.24	0.76	0.2	0.8	0.24	0.76	0.2	0.8
4	8	0.21	0.79	0.16	0.84	0.17	0.83	0.19	0.81		
4	9	0.24	0.76	0	1	0.15	0.85	0.17	0.83		
4	10	0.26	0.74	0	1	0.14	0.86	0.15	0.85		
4	11	0.29	0.71	0	1	0.12	0.88	0.14	0.86		
4	12	0.32	0.68	0	1	0.1	0.9	0.12	0.88		
4	13	0.34	0.66	0	1	0.07	0.93	0.11	0.89		
4	14	0.37	0.63	0	1	0.02	0.98	0.09	0.91		
4	15	0.4	0.6	0	1	0	1	0.07	0.93		
5	1	1	0	0.95	0.05	0.99	0.01	1	0		
5	2	0.9	0.1	0.91	0.09	0.91	0.09	1	0	1 & 0.8	0 & 0.2
5	3	0.88	0.12	0.85	0.15	0.83	0.17	1	0		
5	4	0.84	0.16	0.79	0.21	0.74	0.26	0.81	0.19	0.8	0.2
5	5	0.66	0.34	0.63	0.37	0.62	0.38	0.62	0.38		
5	6	0.42	0.58	0.43	0.57	0.43	0.57	0.44	0.56	0.4	0.6
5	7	0.17	0.83	0.22	0.78	0.21	0.79	0.25	0.75		
5	8	0.08	0.92	0.14	0.86	0.1	0.9	0.15	0.85		
5	9	0.1	0.9	0.05	0.95	0.08	0.92	0.1	0.9		
5	10	0.13	0.87	0	1	0.06	0.94	0.08	0.92		
5	11	0.16	0.84	0	1	0.04	0.96	0.07	0.93		
5	12	0.19	0.81	0	1	0.03	0.97	0.05	0.95		
5	13	0.21	0.79	0	1	0	1	0.04	0.96		
5	14	0.24	0.76	0	1	0	1	0.02	0.98		
5	15	0.27	0.73	0	1	0	1	0.01	0.99		

Table of Results for Transactions INVAR Networks

The selected model has a 3*4*2 architecture :



Bounded INVAR Models

Artificial data was added to the INVAR training set, as shown below:

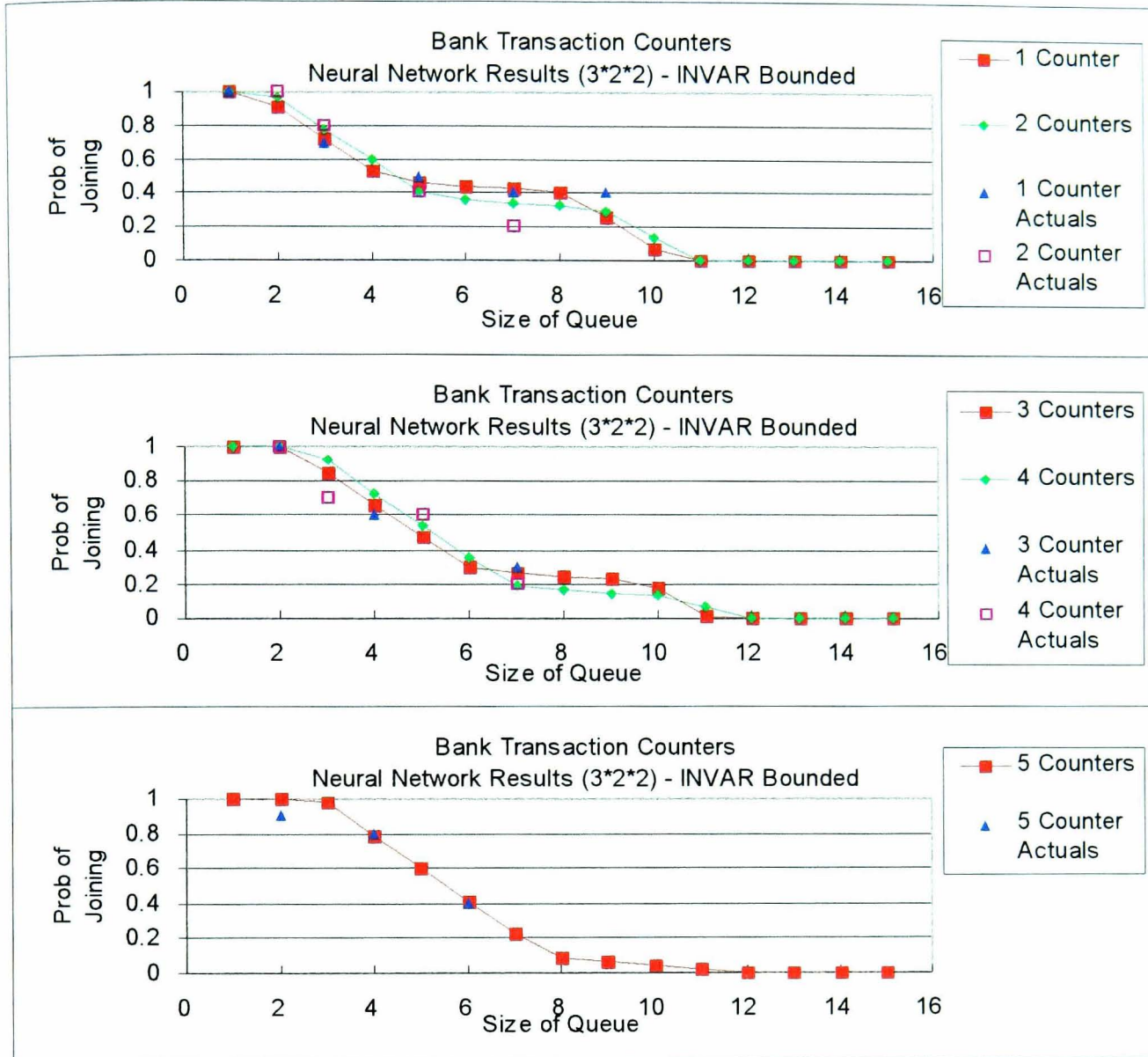
Number Counters	Shortest Length	Probability	Stay Decision	Balk Decision
1	10	0.05	0	1
1	10	0.15	0	1
1	10	0.25	0	1
1	10	0.35	0	1
1	10	0.45	0	1
1	10	0.55	0	1
1	10	0.65	0	1
1	10	0.75	0	1
1	10	0.85	0	1
1	10	0.95	0	1

The results from the Bounded INVAR data sets are :

Counters Open	Queue Length	3*4*2		3*3*2		3*2*2		3*1*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	1	0	1	0	1	0	1	0	1	0
1	2	0.88	0.12	0.88	0.12	0.91	0.09	0.93	0.07		
1	3	0.71	0.29	0.73	0.27	0.72	0.28	0.79	0.21	0.7	0.3
1	4	0.55	0.45	0.6	0.4	0.53	0.47	0.66	0.34		
1	5	0.46	0.54	0.48	0.52	0.46	0.54	0.53	0.47	0.5	0.5
1	6	0.45	0.55	0.43	0.57	0.44	0.56	0.39	0.61		
1	7	0.44	0.56	0.4	0.6	0.43	0.57	0.26	0.74	0.4	0.6
1	8	0.43	0.57	0.33	0.67	0.4	0.6	0.13	0.87		
1	9	0.41	0.59	0.19	0.81	0.26	0.74	0	1	0.4	0.6
1	10	0	1	0.06	0.94	0.07	0.93	0	1	0	1
1	11	0	1	0	1	0	1	0	1		
1	12	0	1	0	1	0	1	0	1		
1	13	0	1	0	1	0	1	0	1		
1	14	0	1	0	1	0	1	0	1		
1	15	0	1	0	1	0	1	0	1		
2	1	1	0	1	0	1	0	1	0		
2	2	0.98	0.02	0.98	0.02	0.97	0.03	0.93	0.07	1	0
2	3	0.81	0.19	0.75	0.25	0.78	0.22	0.8	0.2	0.8	0.2
2	4	0.64	0.36	0.57	0.43	0.59	0.41	0.66	0.34		
2	5	0.43	0.57	0.43	0.57	0.41	0.59	0.53	0.47	0.4	0.6
2	6	0.35	0.65	0.36	0.64	0.36	0.64	0.4	0.6		
2	7	0.2	0.8	0.33	0.67	0.34	0.66	0.26	0.74	0.2	0.8
2	8	0.03	0.97	0.28	0.72	0.33	0.67	0.13	0.87		
2	9	0	1	0.16	0.84	0.29	0.71	0	1		
2	10	0	1	0.02	0.98	0.13	0.87	0	1		
2	11	0	1	0	1	0	1	0	1		
2	12	0	1	0	1	0	1	0	1		
2	13	0	1	0	1	0	1	0	1		
2	14	0	1	0	1	0	1	0	1		
2	15	0	1	0	1	0	1	0	1		
3	1	1	0	1	0	1	0	1	0		
3	2	1	0	1	0	1	0	0.93	0.07	1	0
3	3	0.76	0.24	0.85	0.15	0.85	0.15	0.8	0.2		
3	4	0.57	0.43	0.65	0.35	0.66	0.34	0.67	0.33	0.6	0.4
3	5	0.41	0.59	0.43	0.57	0.47	0.53	0.53	0.47		
3	6	0.31	0.69	0.3	0.7	0.3	0.7	0.4	0.6		
3	7	0.26	0.74	0.27	0.73	0.26	0.74	0.27	0.73	0.3	0.7
3	8	0.12	0.88	0.23	0.77	0.24	0.76	0.13	0.87		
3	9	0	1	0.12	0.88	0.23	0.77	0	1		
3	10	0	1	0	1	0.18	0.82	0	1		
3	11	0	1	0	1	0.01	0.99	0	1		
3	12	0	1	0	1	0	1	0	1		
3	13	0	1	0	1	0	1	0	1		
3	14	0	1	0	1	0	1	0	1		
3	15	0	1	0	1	0	1	0	1		
4	1	1	0	1	0	1	0	1	0		
4	2	0.97	0.03	0.97	0.03	1	0	0.93	0.07	1	0
4	3	0.83	0.17	0.84	0.16	0.92	0.08	0.8	0.2	0.7	0.3
4	4	0.67	0.33	0.7	0.3	0.73	0.27	0.67	0.33		
4	5	0.5	0.5	0.53	0.47	0.54	0.46	0.54	0.46	0.6	0.4
4	6	0.34	0.66	0.32	0.68	0.35	0.65	0.4	0.6		
4	7	0.24	0.76	0.2	0.8	0.19	0.81	0.27	0.73	0.2	0.8
4	8	0.19	0.81	0.16	0.84	0.16	0.84	0.14	0.86		
4	9	0.05	0.95	0.08	0.92	0.14	0.86	0	1		
4	10	0	1	0	1	0.13	0.87	0	1		
4	11	0	1	0	1	0.07	0.93	0	1		
4	12	0	1	0	1	0	1	0	1		
4	13	0	1	0	1	0	1	0	1		
4	14	0	1	0	1	0	1	0	1		
4	15	0	1	0	1	0	1	0	1		
5	1	0.87	0.13	1	0	1	0	1	0		
5	2	0.9	0.1	0.93	0.07	1	0	0.94	0.06	1 & 0.8	0 & 0.2
5	3	0.88	0.12	0.8	0.2	0.98	0.02	0.8	0.2		
5	4	0.77	0.23	0.67	0.33	0.79	0.21	0.67	0.33	0.8	0.2
5	5	0.6	0.4	0.53	0.47	0.6	0.4	0.54	0.46		
5	6	0.43	0.57	0.39	0.61	0.41	0.59	0.4	0.6	0.4	0.6
5	7	0.27	0.73	0.21	0.79	0.23	0.77	0.27	0.73		
5	8	0.18	0.82	0.12	0.88	0.09	0.91	0.14	0.86		
5	9	0.13	0.87	0.06	0.94	0.06	0.94	0	1		
5	10	0	1	0	1	0.04	0.96	0	1		
5	11	0	1	0	1	0.02	0.98	0	1		
5	12	0	1	0	1	0	1	0	1		
5	13	0	1	0	1	0	1	0	1		
5	14	0	1	0	1	0	1	0	1		
5	15	0	1	0	1	0	1	0	1		

Table of Results for Bounded Transactions INVAR Networks

The selected bounded INVAR network is $3*2*2$, shown below :



OUTDATA Models

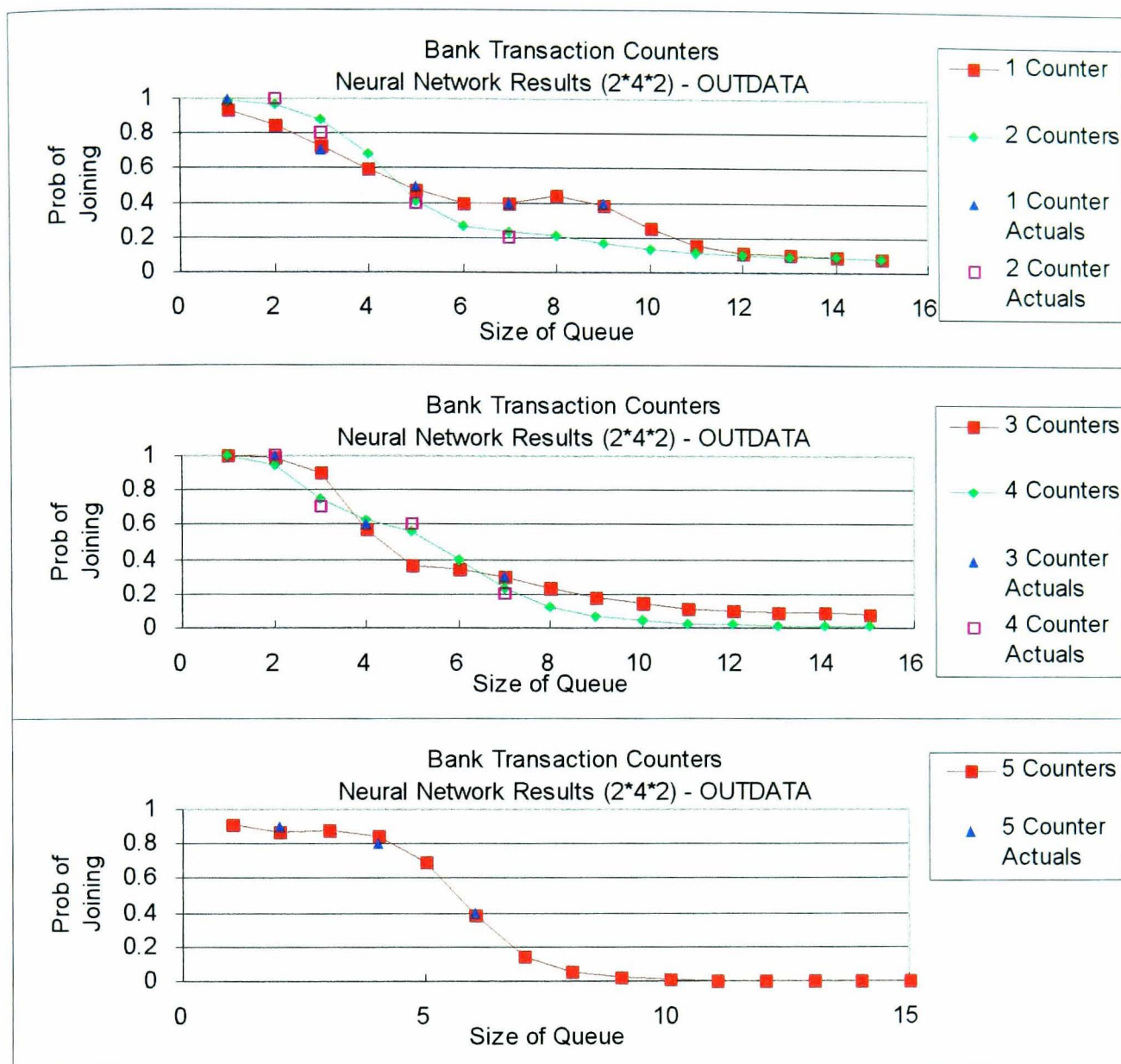
Number of Counters	Shortest Length	Proportion Staying	Proportion Leaving
1	1	1	0
1	3	0.7	0.3
1	5	0.5	0.5
1	7	0.4	0.6
1	9	0.4	0.6
2	2	1	0
2	3	0.8	0.2
2	5	0.4	0.6
2	7	0.2	0.8
3	2	1	0
3	2	1	0
3	4	0.6	0.4
3	7	0.3	0.7
4	2	1	0
4	3	0.7	0.3
4	5	0.6	0.4
4	7	0.2	0.8
5	2	1	0
5	2	0.8	0.2
5	4	0.8	0.2
5	6	0.4	0.6

The results for the OUTDATA models, with different sizes of hidden layer are:

Counters Open	Queue Length	3*6*2		3*4*2		3*3*2		3*2*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	1	0	0.93094	0.06906	1	0	0.95221	0.04779	1	0
1	2	0.99887	0.00113	0.84567	0.15433	1	0	0.88044	0.11956		
1	3	0.74867	0.2516	0.72491	0.27509	0.74192	0.25808	0.73575	0.26425	0.7	0.3
1	4	0.5823	0.41803	0.59228	0.40772	0.61417	0.38583	0.58172	0.41828		
1	5	0.49848	0.50184	0.47382	0.52618	0.48488	0.51512	0.48272	0.51728	0.5	0.5
1	6	0.46117	0.53913	0.3974	0.6026	0.42326	0.57674	0.43073	0.56927		
1	7	0.40358	0.59669	0.39584	0.60416	0.40734	0.59266	0.40322	0.59678	0.4	0.6
1	8	0.35903	0.64121	0.43943	0.56057	0.40419	0.59581	0.38677	0.61323		
1	9	0.40241	0.5978	0.38984	0.61016	0.4037	0.5963	0.37513	0.62487	0.4	0.6
1	10	0.33215	0.66799	0.25604	0.74396	0.40367	0.59633	0.36559	0.63441		
1	11	0.0813	0.91872	0.15929	0.84071	0.40369	0.59631	0.35701	0.64299		
1	12	0.02261	0.97739	0.11366	0.88634	0.40371	0.59629	0.3489	0.6511		
1	13	0.01315	0.98685	0.09358	0.90642	0.40371	0.59629	0.34105	0.65895		
1	14	0.01101	0.98899	0.08441	0.91559	0.40372	0.59628	0.3334	0.6666		
1	15	0.01043	0.98957	0.08	0.92	0.40372	0.59628	0.3259	0.6741		
2	1	1	0	0.99096	0.00904	1	0	0.95482	0.04518		
2	2	0.99545	0.00455	0.9651	0.0349	0.99989	0.00011	0.90838	0.09162	1	0
2	3	0.73142	0.26885	0.88192	0.11808	0.73622	0.26378	0.78465	0.21535	0.8	0.2
2	4	0.57816	0.42216	0.67814	0.32186	0.61021	0.38979	0.58334	0.41666		
2	5	0.39899	0.60131	0.40728	0.59272	0.40693	0.59307	0.41753	0.58247	0.4	0.6
2	6	0.23689	0.76332	0.2637	0.7363	0.26918	0.73082	0.32907	0.67093		
2	7	0.20126	0.79892	0.23482	0.76518	0.2266	0.7734	0.28703	0.71297	0.2	0.8
2	8	0.35707	0.64313	0.20694	0.79306	0.2325	0.7675	0.26594	0.73406		
2	9	0.33192	0.66822	0.16171	0.83829	0.27392	0.72608	0.25374	0.74626		
2	10	0.07659	0.92343	0.12658	0.87342	0.33368	0.66632	0.24534	0.75466		
2	11	0.0218	0.9782	0.10566	0.89434	0.37604	0.62396	0.23859	0.76141		
2	12	0.01299	0.98701	0.09379	0.90621	0.39444	0.60556	0.2326	0.7674		
2	13	0.01097	0.98903	0.08691	0.91309	0.4008	0.5992	0.22701	0.77299		
2	14	0.01042	0.98958	0.08274	0.91726	0.40282	0.59718	0.22166	0.77834		
2	15	0.01026	0.98974	0.08011	0.91989	0.40344	0.59656	0.21648	0.78352		
3	1	1	0	0.9988	0.0012	1	0	0.94791	0.05209		
3	2	0.98539	0.01463	0.9895	0.0105	0.99704	0.00296	0.91969	0.08031	1	0
3	3	0.72555	0.27472	0.90275	0.09725	0.75092	0.24908	0.84088	0.15912		
3	4	0.6079	0.39241	0.57679	0.42321	0.66655	0.33345	0.65936	0.34064	0.6	0.4
3	5	0.44908	0.55122	0.36316	0.63684	0.48313	0.51687	0.43102	0.56898		
3	6	0.3107	0.68954	0.33569	0.66431	0.3037	0.6963	0.28467	0.71533		
3	7	0.30426	0.69594	0.29991	0.70009	0.22751	0.77249	0.21738	0.78262	0.3	0.7
3	8	0.13241	0.86766	0.23409	0.76591	0.20459	0.79541	0.18761	0.81239		
3	9	0.01977	0.98023	0.17701	0.82299	0.19845	0.80155	0.17325	0.82675		
3	10	0.00851	0.99149	0.1389	0.8611	0.1978	0.8022	0.16522	0.83478		
3	11	0.00922	0.99078	0.11506	0.88494	0.20098	0.79902	0.15983	0.84017		
3	12	0.01011	0.98989	0.10008	0.89992	0.21209	0.78791	0.15558	0.84442		
3	13	0.01025	0.98975	0.09042	0.90958	0.24195	0.75805	0.15187	0.84813		
3	14	0.01023	0.98977	0.08404	0.91596	0.29763	0.70237	0.14843	0.85157		
3	15	0.01021	0.98979	0.07971	0.92029	0.35438	0.64562	0.14516	0.85484		
4	1	1	0	0.99514	0.00486	1	0	0.93305	0.06695		
4	2	0.96356	0.03648	0.94209	0.05791	0.97342	0.02658	0.91564	0.08436	1	0
4	3	0.72237	0.2779	0.75177	0.24823	0.76151	0.23849	0.87076	0.12924	0.7	0.3
4	4	0.64669	0.35359	0.62751	0.37249	0.70704	0.29296	0.75367	0.24633		
4	5	0.60862	0.39164	0.55608	0.44392	0.56161	0.43839	0.52813	0.47187	0.6	0.4
4	6	0.59763	0.40259	0.40076	0.59924	0.35844	0.64156	0.30879	0.69121		
4	7	0.20511	0.79499	0.22713	0.77287	0.24641	0.75359	0.1941	0.8059	0.2	0.8
4	8	0.01886	0.98114	0.11799	0.88201	0.20989	0.79011	0.14637	0.85363		
4	9	0.00431	0.99569	0.0647	0.9353	0.19947	0.80053	0.12616	0.87384		
4	10	0.0025	0.9975	0.03945	0.96055	0.19657	0.80343	0.11671	0.88329		
4	11	0.0023	0.9977	0.02677	0.97323	0.19578	0.80422	0.11157	0.88843		
4	12	0.00313	0.99687	0.01987	0.98013	0.19561	0.80439	0.10823	0.89177		
4	13	0.00588	0.99412	0.01582	0.98418	0.19573	0.80427	0.10565	0.89435		
4	14	0.00878	0.99122	0.01329	0.98671	0.19629	0.80371	0.10343	0.89657		
4	15	0.00988	0.99013	0.01161	0.98839	0.19815	0.80185	0.10139	0.89861		
5	1	0.99999	0.00001	0.91528	0.08472	1	0	0.91205	0.08795		
5	2	0.92958	0.07049	0.87222	0.12778	0.91508	0.08492	0.90119	0.09881	1 & 0.8	0 & 0.2
5	3	0.74414	0.2561	0.88342	0.11658	0.76863	0.23137	0.87587	0.12413		
5	4	0.7961	0.20408	0.85114	0.14886	0.73415	0.26585	0.81192	0.18808	0.8	0.2
5	5	0.84123	0.15889	0.69036	0.30964	0.6294	0.3706	0.65872	0.34128		
5	6	0.40405	0.5961	0.38609	0.61391	0.429	0.571	0.41384	0.58616	0.4	0.6
5	7	0.03303	0.96698	0.14717	0.85283	0.27567	0.72433	0.22341	0.77659		
5	8	0.00571	0.99429	0.0507	0.9493	0.21864	0.78136	0.13727	0.86273		
5	9	0.00276	0.99724	0.01939	0.98061	0.20192	0.79808	0.1035	0.8965		
5	10	0.00213	0.99787	0.00877	0.99123	0.19724	0.80276	0.08959	0.91041		
5	11	0.00195	0.99805	0.00469	0.99531	0.19594	0.80406	0.08322	0.91678		
5	12	0.00191	0.99809	0.00291	0.99709	0.19558	0.80442	0.07987	0.92013		
5	13	0.002	0.998	0.00203	0.99797	0.19549	0.80451	0.07776	0.92224		
5	14	0.0025	0.9975	0.00156	0.99844	0.19546	0.80454	0.07618	0.92382		
5	15	0.00446	0.99554	0.00128	0.99872	0.19546	0.80454	0.07484	0.92516		

Table of Results for Transactions OUTDATA Networks

The selected OUTDATA model has a 2*4*2 architecture :



OUTNET Models

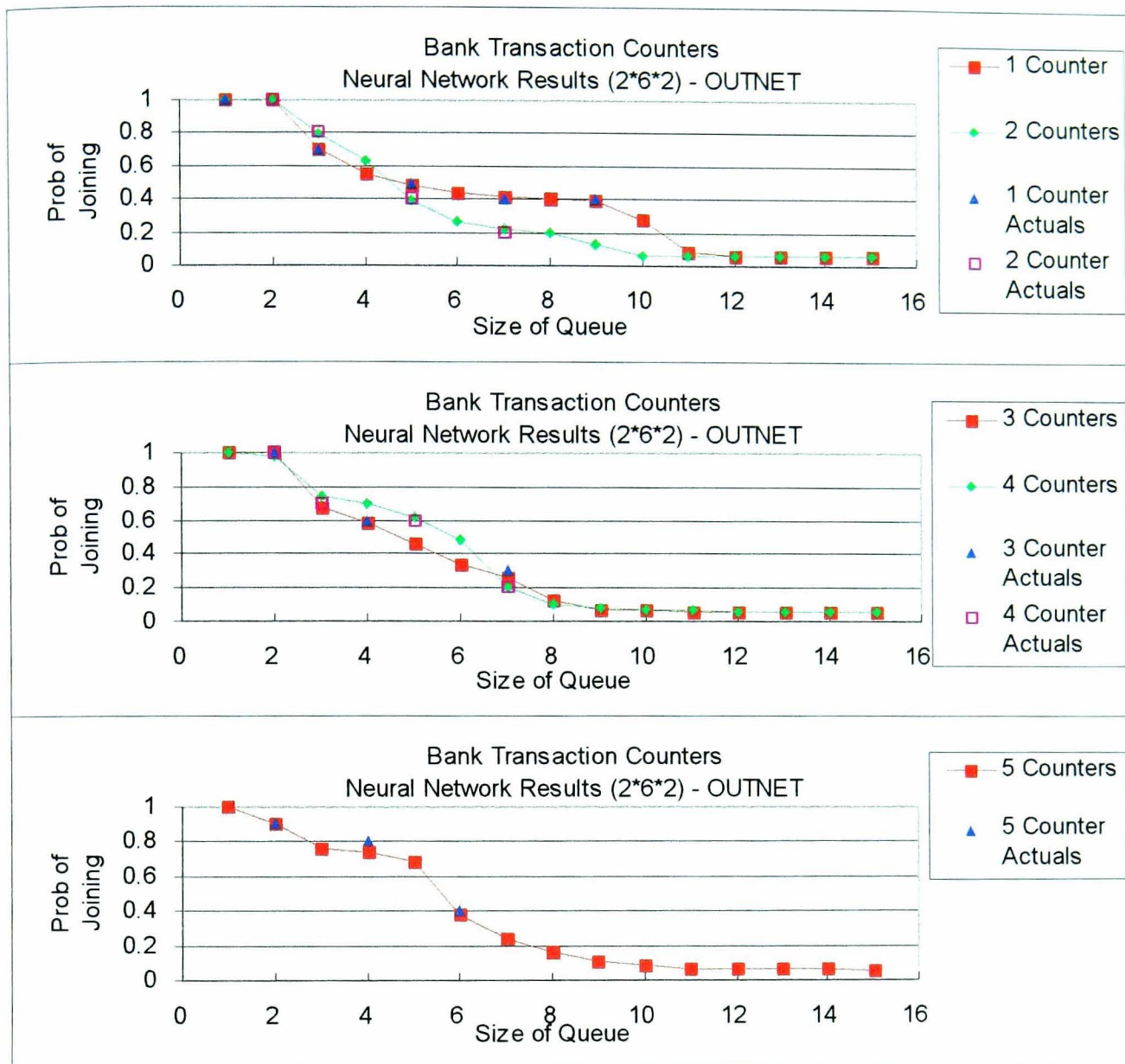
The OUTNET models were training with the same data as the INVAR models, except that the probability input variable is not used (see page H-13).

The results for the OUTNET models, with different sizes of hidden layer are:

Counters Open	Queue Length	2*8*2		2*6*2		2*4*2		2*2*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	1	0	1	0	1	0	0.94165	0.05835	1	0
1	2	1	0	1	0	1	0	0.87829	0.12171		
1	3	0.69996	0.30004	0.70252	0.29747	0.7057	0.2943	0.69853	0.30147	0.7	0.3
1	4	0.41338	0.58662	0.54996	0.45003	0.50343	0.49657	0.51992	0.48008		
1	5	0.49876	0.50124	0.47861	0.52139	0.49973	0.50027	0.44704	0.55296	0.5	0.5
1	6	0.59381	0.40619	0.43801	0.56199	0.51799	0.48201	0.42544	0.57456		
1	7	0.39998	0.60002	0.41692	0.58307	0.41189	0.58811	0.41872	0.58128	0.4	0.6
1	8	0.3841	0.6159	0.40512	0.59488	0.36862	0.63138	0.41031	0.58969		
1	9	0.39979	0.60021	0.38823	0.61177	0.39944	0.60056	0.36548	0.63452	0.4	0.6
1	10	0.49249	0.50751	0.27962	0.72038	0.4337	0.5663	0.26281	0.73719		
1	11	0.70496	0.29504	0.08563	0.91437	0.46906	0.53094	0.21879	0.78121		
1	12	0.76061	0.23939	0.06233	0.93767	0.50345	0.49655	0.21208	0.78792		
1	13	0.76586	0.23414	0.05951	0.94049	0.53508	0.46492	0.21126	0.78874		
1	14	0.76632	0.23368	0.05911	0.94089	0.56271	0.43729	0.21116	0.78884		
1	15	0.76636	0.23364	0.05904	0.94096	0.58584	0.41416	0.21115	0.78885		
2	1	1	0	1	0	1	0	0.95443	0.04557		
2	2	0.99998	0.00002	1	0	1	0	0.9313	0.0687	1	0
2	3	0.79846	0.20154	0.78783	0.21216	0.83532	0.16468	0.8387	0.1613	0.8	0.2
2	4	0.45492	0.54508	0.62818	0.37181	0.56646	0.43354	0.634	0.366		
2	5	0.3991	0.6009	0.394	0.606	0.38537	0.61463	0.46828	0.53172	0.4	0.6
2	6	0.34496	0.65504	0.26149	0.73851	0.24498	0.75502	0.33341	0.66659		
2	7	0.19865	0.80135	0.21599	0.78401	0.24232	0.75768	0.23656	0.76344	0.2	0.8
2	8	0.3732	0.6268	0.19557	0.80442	0.24239	0.75761	0.21491	0.78509		
2	9	0.67745	0.32255	0.13181	0.86819	0.24255	0.75745	0.21175	0.78825		
2	10	0.75611	0.24389	0.06314	0.93686	0.24278	0.75722	0.21126	0.78874		
2	11	0.76491	0.23509	0.0597	0.9403	0.24308	0.75692	0.21117	0.78883		
2	12	0.76592	0.23408	0.05932	0.94068	0.24349	0.75651	0.21116	0.78884		
2	13	0.76599	0.23401	0.05916	0.94084	0.24406	0.75594	0.21115	0.78885		
2	14	0.76591	0.23409	0.05909	0.94091	0.24484	0.75516	0.21115	0.78885		
2	15	0.76577	0.23423	0.05905	0.94095	0.24589	0.75411	0.21115	0.78885		
3	1	1	0	1	0	1	0	0.95794	0.04206		
3	2	0.99952	0.00048	0.99959	0.00041	0.99985	0.00015	0.93978	0.06022	1	0
3	3	0.8025	0.1975	0.67808	0.32192	0.86278	0.13722	0.84487	0.15513		
3	4	0.60006	0.39994	0.58179	0.4182	0.63606	0.36394	0.59184	0.40816	0.6	0.4
3	5	0.4052	0.5948	0.45854	0.54146	0.30618	0.69382	0.34289	0.65711		
3	6	0.39126	0.60874	0.34222	0.65777	0.24616	0.75384	0.24734	0.75266		
3	7	0.29872	0.70128	0.25925	0.74075	0.24221	0.75779	0.22081	0.77919	0.3	0.7
3	8	0.35917	0.64083	0.1292	0.8708	0.24197	0.75803	0.21373	0.78627		
3	9	0.53833	0.46167	0.06781	0.93219	0.24196	0.75804	0.21184	0.78816		
3	10	0.69461	0.30539	0.06249	0.93751	0.24196	0.75804	0.21133	0.78867		
3	11	0.7388	0.2612	0.06065	0.93935	0.24196	0.75804	0.2112	0.7888		
3	12	0.74426	0.25574	0.05979	0.94021	0.24196	0.75804	0.21116	0.78884		
3	13	0.74068	0.25932	0.05939	0.94061	0.24196	0.75804	0.21115	0.78885		
3	14	0.73421	0.26579	0.05919	0.94081	0.24196	0.75804	0.21115	0.78885		
3	15	0.72581	0.27419	0.0591	0.9409	0.24196	0.75804	0.21115	0.78885		
4	1	1	0	1	0	1	0	0.90337	0.09663		
4	2	0.98945	0.01055	0.98147	0.01852	0.98745	0.01255	0.89455	0.10545	1	0
4	3	0.69992	0.30008	0.73637	0.26362	0.74718	0.25282	0.86529	0.13471	0.7	0.3
4	4	0.68117	0.31883	0.69154	0.30845	0.72919	0.27081	0.75528	0.24472		
4	5	0.5992	0.4008	0.61244	0.38755	0.53786	0.46214	0.49845	0.50155	0.6	0.4
4	6	0.51045	0.48955	0.48221	0.51779	0.27514	0.72486	0.30184	0.69816		
4	7	0.20009	0.79991	0.19889	0.80111	0.24405	0.75595	0.23577	0.76423	0.2	0.8
4	8	0.14211	0.85789	0.10123	0.89877	0.24208	0.75792	0.21773	0.78227		
4	9	0.14386	0.85614	0.07843	0.92157	0.24196	0.75804	0.21291	0.78709		
4	10	0.19319	0.80681	0.06813	0.93187	0.24195	0.75805	0.21162	0.78838		
4	11	0.33675	0.66325	0.06331	0.93669	0.24195	0.75805	0.21128	0.78872		
4	12	0.44672	0.55328	0.06104	0.93896	0.24195	0.75805	0.21118	0.78882		
4	13	0.47188	0.52812	0.05998	0.94002	0.24195	0.75805	0.21116	0.78884		
4	14	0.47076	0.52924	0.05947	0.94053	0.24195	0.75805	0.21115	0.78885		
4	15	0.46537	0.53463	0.05924	0.94076	0.24195	0.75805	0.21115	0.78885		
5	1	1	0	1	0	1	0	0.90351	0.09649		
5	2	0.89789	0.10211	0.9008	0.0992	0.90335	0.09665	0.9006	0.0994	1 & 0.8	0 & 0.2
5	3	0.75478	0.24522	0.76088	0.23911	0.74736	0.25264	0.88951	0.11049		
5	4	0.79858	0.20142	0.74238	0.25762	0.745	0.255	0.84626	0.15374	0.8	0.2
5	5	0.68714	0.31286	0.68392	0.31608	0.71163	0.28837	0.69292	0.30708		
5	6	0.39697	0.60303	0.38563	0.61437	0.44174	0.55826	0.42429	0.57571	0.4	0.6
5	7	0.13493	0.86507	0.24451	0.75549	0.25875	0.74125	0.27358	0.72642		
5	8	0.09411	0.90589	0.16219	0.83781	0.24299	0.75701	0.22794	0.77206		
5	9	0.09286	0.90714	0.11016	0.88984	0.24202	0.75798	0.21563	0.78437		
5	10	0.0959	0.9041	0.08323	0.91677	0.24196	0.75804	0.21235	0.78765		
5	11	0.11267	0.88733	0.07039	0.92961	0.24195	0.75805	0.21147	0.78853		
5	12	0.1868	0.8132	0.06437	0.93563	0.24195	0.75805	0.21124	0.78876		
5	13	0.33435	0.66565	0.06154	0.93846	0.24195	0.75805	0.21117	0.78883		
5	14	0.41576	0.58424	0.06021	0.93979	0.24195	0.75805	0.21116	0.78884		
5	15	0.43594	0.56406	0.05958	0.94042	0.24195	0.75805	0.21115	0.78885		

Table of Results for Transactions OUTNET Networks

The chosen OUTNET network was a $2*6*2$ network :



H3 : Business Service

The training data and results from models are shown for the Stay/Balk decision involving customers of the Business Service.

INVAR Models

The unscaled training data for the INVAR models is shown below.

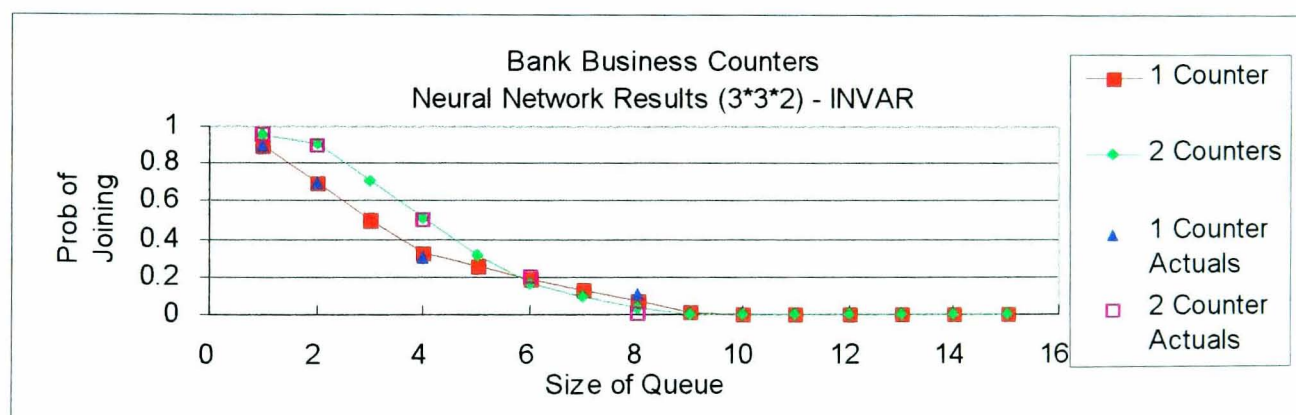
Number Counters	Shortest Length	Probability	Stay Decision	Balk Decision	Number Counters	Shortest Queue	Probability	Stay Decision	Balk Decision
1	1	0.05	1	0	2	1	0.525	1	0
1	1	0.15	1	0	2	1	0.575	1	0
1	1	0.25	1	0	2	1	0.625	1	0
1	1	0.35	1	0	2	1	0.675	1	0
1	1	0.45	1	0	2	1	0.725	1	0
1	1	0.55	1	0	2	1	0.775	1	0
1	1	0.65	1	0	2	1	0.825	1	0
1	1	0.75	1	0	2	1	0.875	1	0
1	1	0.85	1	0	2	1	0.925	1	0
1	1	0.95	0	1	2	1	0.975	0	1
1	2	0.05	1	0	2	2	0.05	1	0
1	2	0.15	1	0	2	2	0.15	1	0
1	2	0.25	1	0	2	2	0.25	1	0
1	2	0.35	1	0	2	2	0.35	1	0
1	2	0.45	1	0	2	2	0.45	1	0
1	2	0.55	1	0	2	2	0.55	1	0
1	2	0.65	1	0	2	2	0.65	1	0
1	2	0.75	0	1	2	2	0.75	1	0
1	2	0.85	0	1	2	2	0.85	1	0
1	2	0.95	0	1	2	2	0.95	0	1
1	4	0.05	1	0	2	4	0.05	1	0
1	4	0.15	1	0	2	4	0.15	1	0
1	4	0.25	1	0	2	4	0.25	1	0
1	4	0.35	0	1	2	4	0.35	1	0
1	4	0.45	0	1	2	4	0.45	1	0
1	4	0.55	0	1	2	4	0.55	0	1
1	4	0.65	0	1	2	4	0.65	0	1
1	4	0.75	0	1	2	4	0.75	0	1
1	4	0.85	0	1	2	4	0.85	0	1
1	4	0.95	0	1	2	4	0.95	0	1
1	8	0.05	1	0	2	6	0.05	1	0
1	8	0.15	0	1	2	6	0.15	1	0
1	8	0.25	0	1	2	6	0.25	0	1
1	8	0.35	0	1	2	6	0.35	0	1
1	8	0.45	0	1	2	6	0.45	0	1
1	8	0.55	0	1	2	6	0.55	0	1
1	8	0.65	0	1	2	6	0.65	0	1
1	8	0.75	0	1	2	6	0.75	0	1
1	8	0.85	0	1	2	6	0.85	0	1
1	8	0.95	0	1	2	6	0.95	0	1
2	1	0.025	1	0	2	8	0.05	0	1
2	1	0.075	1	0	2	8	0.15	0	1
2	1	0.125	1	0	2	8	0.25	0	1
2	1	0.175	1	0	2	8	0.35	0	1
2	1	0.225	1	0	2	8	0.45	0	1
2	1	0.275	1	0	2	8	0.55	0	1
2	1	0.325	1	0	2	8	0.65	0	1
2	1	0.375	1	0	2	8	0.75	0	1
2	1	0.425	1	0	2	8	0.85	0	1
2	1	0.475	1	0	2	8	0.95	0	1

The results for the INVAR models are :

Counters Open	Queue Length	2*6*2		2*4*2		2*3*2		2*2*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	0.9	0.1	0.89	0.11	0.89	0.11	0.87	0.13	0.9	0.1
1	2	0.71	0.29	0.71	0.29	0.7	0.3	0.69	0.31	0.7	0.3
1	3	0.46	0.54	0.46	0.54	0.5	0.5	0.51	0.49		
1	4	0.28	0.72	0.27	0.73	0.33	0.67	0.33	0.67	0.3	0.7
1	5	0.26	0.74	0.25	0.75	0.26	0.74	0.15	0.85		
1	6	0.25	0.75	0.23	0.77	0.19	0.81	0	1		
1	7	0.2	0.8	0.2	0.8	0.13	0.87	0	1		
1	8	0.1	0.9	0.1	0.9	0.07	0.93	0	1	0.1	0.9
1	9	0	1	0	1	0.01	0.99	0	1		
1	10	0	1	0	1	0	1	0	1		
1	11	0	1	0	1	0	1	0	1		
1	12	0	1	0	1	0	1	0	1		
1	13	0	1	0	1	0	1	0	1		
1	14	0	1	0	1	0	1	0	1		
1	15	0	1	0	1	0	1	0	1		
2	1	0.95	0.05	0.95	0.05	0.95	0.05	0.95	0.05	1	0
2	2	0.92	0.08	0.91	0.09	0.91	0.09	0.88	0.12	0.9	0.1
2	3	0.72	0.28	0.72	0.28	0.71	0.29	0.71	0.29		
2	4	0.5	0.5	0.48	0.52	0.51	0.49	0.53	0.47	0.5	0.5
2	5	0.32	0.68	0.33	0.67	0.31	0.69	0.35	0.65		
2	6	0.2	0.8	0.2	0.8	0.16	0.84	0.17	0.83	0.2	0.8
2	7	0.09	0.91	0.09	0.91	0.09	0.91	0	1		
2	8	0	1	0	1	0.03	0.97	0	1	0	1
2	9	0	1	0	1	0	1	0	1		
2	10	0	1	0	1	0	1	0	1		
2	11	0	1	0	1	0	1	0	1		
2	12	0	1	0	1	0	1	0	1		
2	13	0	1	0	1	0	1	0	1		
2	14	0	1	0	1	0	1	0	1		
2	15	0	1	0	1	0	1	0	1		

Table of Results for Business INVAR Models

The selected model has a 3*3*2 architecture :



OUTDATA Models

The training data for the OUTDATA models is :

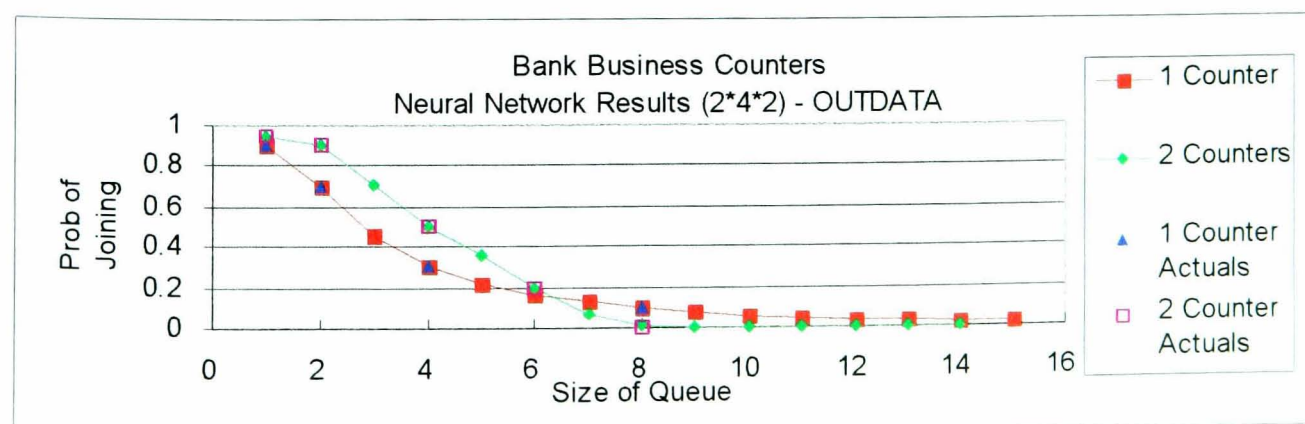
Number of Counters	Shortest Length	Probability of Staying	Probability of Leaving
1	1	0.9	0.1
1	2	0.7	0.3
1	4	0.3	0.7
1	8	0.1	0.9
2	1	0.95	0.05
2	2	0.9	0.1
2	4	0.5	0.5
2	6	0.2	0.8
2	8	0	1

The results from the OUTDATA models are :

Counters Open	Queue Length	2*6*2		2*4*2		2*3*2		2*2*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	0.900086	0.099914	0.905	0.095	0.90573	0.09427	0.87933	0.12067	0.9	0.1
1	2	0.699745	0.300255	0.69483	0.30517	0.69572	0.30428	0.71025	0.28975	0.7	0.3
1	3	0.472514	0.527486	0.45156	0.54844	0.45263	0.54737	0.48902	0.51098		
1	4	0.299899	0.700101	0.3001	0.6999	0.30102	0.69898	0.30957	0.69043	0.3	0.7
1	5	0.196534	0.803466	0.2175	0.7825	0.21893	0.78107	0.2003	0.7997		
1	6	0.141616	0.858384	0.16614	0.83386	0.16836	0.83164	0.1395	0.8605		
1	7	0.113686	0.886314	0.12873	0.87127	0.13086	0.86914	0.10512	0.89488		
1	8	0.099857	0.900143	0.09927	0.90073	0.09958	0.90042	0.08455	0.91545	0.1	0.9
1	9	0.093509	0.906491	0.07598	0.92402	0.07277	0.92723	0.07142	0.92858		
1	10	0.09127	0.90873	0.05811	0.94189	0.05059	0.94941	0.06248	0.93752		
1	11	0.091333	0.908667	0.04487	0.95513	0.03344	0.96656	0.05605	0.94395		
1	12	0.092727	0.907273	0.03533	0.96467	0.02115	0.97885	0.05118	0.94882		
1	13	0.09488	0.90512	0.02858	0.97142	0.01296	0.98704	0.04735	0.95265		
1	14	0.097453	0.902547	0.02383	0.97617	0.00784	0.99216	0.04423	0.95577		
1	15	0.100247	0.899753	0.02049	0.97951	0.00476	0.99524	0.04163	0.95837		
2	1	0.95012	0.04988	0.95098	0.04902	0.95072	0.04928	0.95465	0.04535	1	0
2	2	0.899759	0.100241	0.8987	0.1013	0.89913	0.10087	0.89473	0.10527	0.9	0.1
2	3	0.706043	0.293957	0.70795	0.29205	0.70046	0.29954	0.75363	0.24637		
2	4	0.49996	0.50004	0.50009	0.49991	0.50048	0.49952	0.51842	0.48158	0.5	0.5
2	5	0.369887	0.630113	0.35386	0.64614	0.35781	0.64219	0.29064	0.70936		
2	6	0.199594	0.800406	0.19642	0.80358	0.19688	0.80312	0.15357	0.84643	0.2	0.8
2	7	0.04492	0.95508	0.06647	0.93353	0.06612	0.93388	0.08772	0.91228		
2	8	0.00442	0.99558	0.01438	0.98562	0.01459	0.98541	0.05694	0.94306	0	1
2	9	0.00055	0.99945	0.00303	0.99697	0.0032	0.9968	0.04177	0.95823		
2	10	0.00017	0.99983	0.00092	0.99908	0.001	0.999	0.03376	0.96624		
2	11	0.0001	0.9999	0.00045	0.99955	0.00049	0.99951	0.02929	0.97071		
2	12	0.00009	0.99991	0.00031	0.99969	0.00033	0.99967	0.02668	0.97332		
2	13	0.00008	0.99992	0.00026	0.99974	0.00027	0.99973	0.0251	0.9749		
2	14	0.00009	0.99991	0.00025	0.99975	0.00024	0.99976	0.02413	0.97587		
2	15	0.00009	0.99991	0.00025	0.99975	0.00023	0.99977	0.02353	0.97647		

Table of Results from Business OUTDATA Models

The chosen model has a 2*4*2 architecture :



OUTNET Models

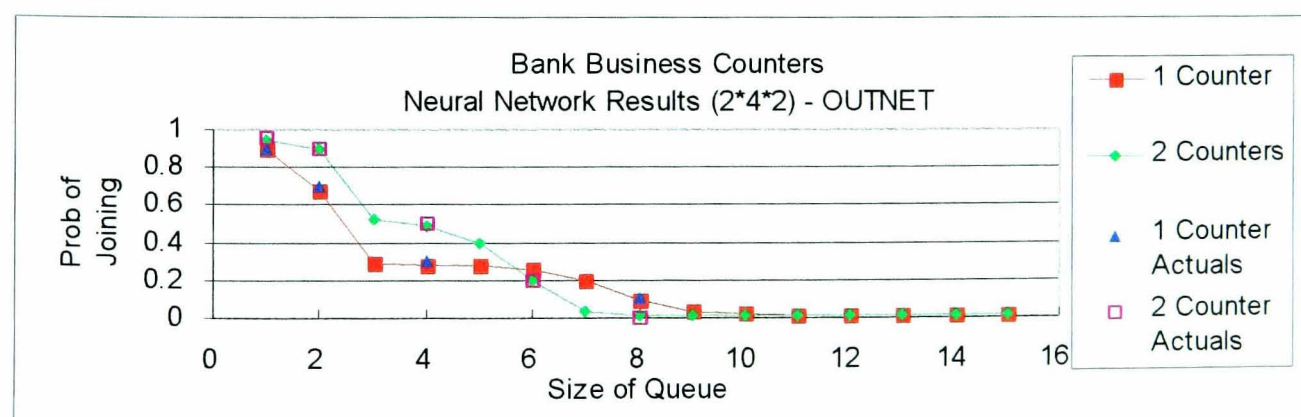
The training data for the OUTNET models is the same as for the INVAR models but with no probability input variable (see page H-24).

The results from the OUTNET models are :

Counters Open	Queue Length	2*6*2		2*4*2		2*3*2		2*2*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	0.896864	0.103136	0.89229	0.10771	0.89413	0.10587	0.91308	0.08692	0.9	0.1
1	2	0.693735	0.306265	0.679977	0.320023	0.68735	0.31265	0.70489	0.29511	0.7	0.3
1	3	0.526714	0.473286	0.295128	0.704872	0.32153	0.67847	0.50626	0.49374		
1	4	0.292315	0.707685	0.283755	0.716245	0.28757	0.71243	0.30677	0.69323	0.3	0.7
1	5	0.155206	0.844794	0.276825	0.723175	0.28413	0.71587	0.06689	0.93311		
1	6	0.146312	0.853688	0.254521	0.745479	0.27353	0.72647	0.05064	0.94936		
1	7	0.139975	0.860025	0.194203	0.805797	0.22057	0.77943	0.04996	0.95004		
1	8	0.095729	0.904271	0.097395	0.902605	0.09216	0.90784	0.04993	0.95007	0.1	0.9
1	9	0.00603	0.99397	0.035806	0.964194	0.02732	0.97268	0.04993	0.95007		
1	10	0.00003	0.99997	0.017837	0.982163	0.01704	0.98296	0.04993	0.95007		
1	11	0	1	0.013414	0.986586	0.0155	0.9845	0.04993	0.95007		
1	12	0	1	0.012244	0.987756	0.01525	0.98475	0.04993	0.95007		
1	13	0	1	0.011914	0.988086	0.01521	0.98479	0.04993	0.95007		
1	14	0	1	0.011824	0.988176	0.0152	0.9848	0.04993	0.95007		
1	15	0	1	0.011803	0.988197	0.0152	0.9848	0.04993	0.95007		
2	1	0.94926	0.05074	0.945478	0.054522	0.94854	0.05146	0.946	0.054	1	0
2	2	0.900376	0.099624	0.891178	0.108822	0.88892	0.11108	0.90529	0.09471	0.9	0.1
2	3	0.588507	0.411493	0.526662	0.473338	0.6675	0.3325	0.68063	0.31937		
2	4	0.49393	0.50607	0.488856	0.511144	0.47406	0.52594	0.50979	0.49021	0.5	0.5
2	5	0.447956	0.552044	0.399379	0.600621	0.31622	0.68378	0.45887	0.54113		
2	6	0.19317	0.80683	0.196797	0.803203	0.18644	0.81356	0.20443	0.79557	0.2	0.8
2	7	0.051503	0.948497	0.037679	0.962321	0.0287	0.9713	0.05726	0.94274		
2	8	0.0005	0.9995	0.012244	0.987756	0.0158	0.9842	0.05024	0.94976	0	1
2	9	0.00001	0.99999	0.011833	0.988167	0.01523	0.98477	0.04994	0.95006		
2	10	0	1	0.011794	0.988206	0.0152	0.9848	0.04993	0.95007		
2	11	0	1	0.011793	0.988207	0.0152	0.9848	0.04993	0.95007		
2	12	0	1	0.011793	0.988207	0.0152	0.9848	0.04993	0.95007		
2	13	0	1	0.011793	0.988207	0.0152	0.9848	0.04993	0.95007		
2	14	0	1	0.011793	0.988207	0.0152	0.9848	0.04993	0.95007		
2	15	0	1	0.011793	0.988207	0.0152	0.9848	0.04993	0.95007		

Table of Results from Business OUTNET Models

The chosen network has a 2*4*2 architecture :



H4 : Currency Service

The training data and results from models are shown for the Stay/Balk decision involving customers of the Currency Service.

INVAR Models

The unscaled training data for the INVAR models is shown below.

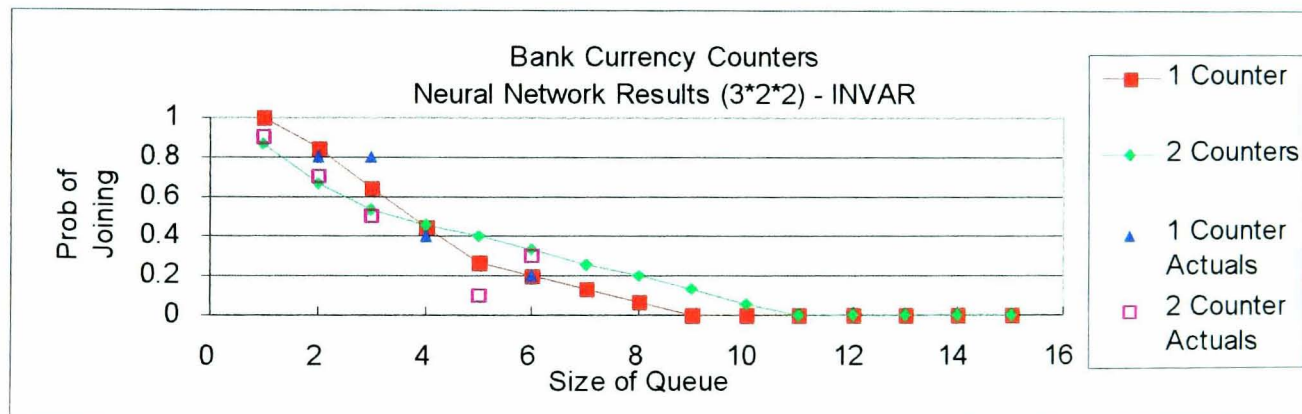
Number Counters	Shortest Length	Probability	Stay Decision	Balk Decision	Number Counters	Shortest Queue	Probability	Stay Decision	Balk Decision
1	2	0.05	1	0	2	2	0.05	1	0
1	2	0.15	1	0	2	2	0.15	1	0
1	2	0.25	1	0	2	2	0.25	1	0
1	2	0.35	1	0	2	2	0.35	1	0
1	2	0.45	1	0	2	2	0.45	1	0
1	2	0.55	1	0	2	2	0.55	1	0
1	2	0.65	1	0	2	2	0.65	1	0
1	2	0.75	1	0	2	2	0.75	0	1
1	2	0.85	0	1	2	2	0.85	0	1
1	2	0.95	0	1	2	2	0.95	0	1
1	3	0.05	1	0	2	3	0.05	1	0
1	3	0.15	1	0	2	3	0.15	1	0
1	3	0.25	1	0	2	3	0.25	1	0
1	3	0.35	1	0	2	3	0.35	1	0
1	3	0.45	1	0	2	3	0.45	1	0
1	3	0.55	1	0	2	3	0.55	0	1
1	3	0.65	1	0	2	3	0.65	0	1
1	3	0.75	1	0	2	3	0.75	0	1
1	3	0.85	0	1	2	3	0.85	0	1
1	3	0.95	0	1	2	3	0.95	0	1
1	4	0.05	1	0	2	4	0.05	1	0
1	4	0.15	1	0	2	4	0.15	1	0
1	4	0.25	1	0	2	4	0.25	1	0
1	4	0.35	1	0	2	4	0.35	1	0
1	4	0.45	0	1	2	4	0.45	1	0
1	4	0.55	0	1	2	4	0.55	0	1
1	4	0.65	0	1	2	4	0.65	0	1
1	4	0.75	0	1	2	4	0.75	0	1
1	4	0.85	0	1	2	4	0.85	0	1
1	4	0.95	0	1	2	4	0.95	0	1
1	6	0.05	1	0	2	5	0.05	1	0
1	6	0.15	1	0	2	5	0.15	0	1
1	6	0.25	0	1	2	5	0.25	0	1
1	6	0.35	0	1	2	5	0.35	0	1
1	6	0.45	0	1	2	5	0.45	0	1
1	6	0.55	0	1	2	5	0.55	0	1
1	6	0.65	0	1	2	5	0.65	0	1
1	6	0.75	0	1	2	5	0.75	0	1
1	6	0.85	0	1	2	5	0.85	0	1
1	6	0.95	0	1	2	5	0.95	0	1
2	1	0.05	1	0	2	6	0.05	1	0
2	1	0.15	1	0	2	6	0.15	1	0
2	1	0.25	1	0	2	6	0.25	1	0
2	1	0.35	1	0	2	6	0.35	0	1
2	1	0.45	1	0	2	6	0.45	0	1
2	1	0.55	1	0	2	6	0.55	0	1
2	1	0.65	1	0	2	6	0.65	0	1
2	1	0.75	1	0	2	6	0.75	0	1
2	1	0.85	1	0	2	6	0.85	0	1
2	1	0.95	0	1	2	6	0.95	0	1

The results for the INVAR models are :

Counters Open	Queue Length	2*4*2		2*3*2		2*2*2		2*1*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	0.95	0.05	1	0	1	0	0.95	0.05		
1	2	0.82	0.18	0.82	0.18	0.85	0.15	0.8	0.2	0.8	0.2
1	3	0.78	0.22	0.78	0.22	0.65	0.35	0.66	0.34	0.8	0.2
1	4	0.4	0.6	0.41	0.59	0.45	0.55	0.51	0.49	0.4	0.6
1	5	0.25	0.75	0.22	0.78	0.27	0.73	0.36	0.64		
1	6	0.24	0.76	0.17	0.83	0.2	0.8	0.21	0.79	0.2	0.8
1	7	0.24	0.76	0.14	0.86	0.13	0.87	0.06	0.94		
1	8	0.24	0.76	0.11	0.89	0.07	0.93	0	1		
1	9	0.23	0.77	0.08	0.92	0	1	0	1		
1	10	0.23	0.77	0.05	0.95	0	1	0	1		
1	11	0.21	0.79	0.02	0.98	0	1	0	1		
1	12	0.19	0.81	0	1	0	1	0	1		
1	13	0.13	0.87	0	1	0	1	0	1		
1	14	0.06	0.94	0	1	0	1	0	1		
1	15	0	1	0	1	0	1	0	1		
2	1	0.89	0.11	0.92	0.08	0.87	0.13	0.87	0.13	0.9	0.1
2	2	0.71	0.29	0.66	0.34	0.67	0.33	0.73	0.27	0.7	0.3
2	3	0.54	0.46	0.55	0.45	0.53	0.47	0.58	0.42	0.5	0.5
2	4	0.48	0.52	0.41	0.59	0.46	0.54	0.43	0.57		
2	5	0.1	0.9	0.11	0.89	0.4	0.6	0.28	0.72	0.1	0.9
2	6	0.29	0.71	0.29	0.71	0.33	0.67	0.13	0.87	0.3	0.7
2	7	0.46	0.54	0.43	0.57	0.26	0.74	0	1		
2	8	0.44	0.56	0.46	0.54	0.2	0.8	0	1		
2	9	0.37	0.63	0.44	0.56	0.13	0.87	0	1		
2	10	0.29	0.71	0.41	0.59	0.06	0.94	0	1		
2	11	0.22	0.78	0.38	0.62	0	1	0	1		
2	12	0.15	0.85	0.35	0.65	0	1	0	1		
2	13	0.07	0.93	0.32	0.68	0	1	0	1		
2	14	0	1	0.29	0.71	0	1	0	1		
2	15	0	1	0.26	0.74	0	1	0	1		

Table of Results for Currency INVAR Models

The selected model has a 3*2*2 architecture :



Bounded INVAR Models

The bounded INVAR models were trained with artificial data added to the INVAR training data. The artificial data is shown below :

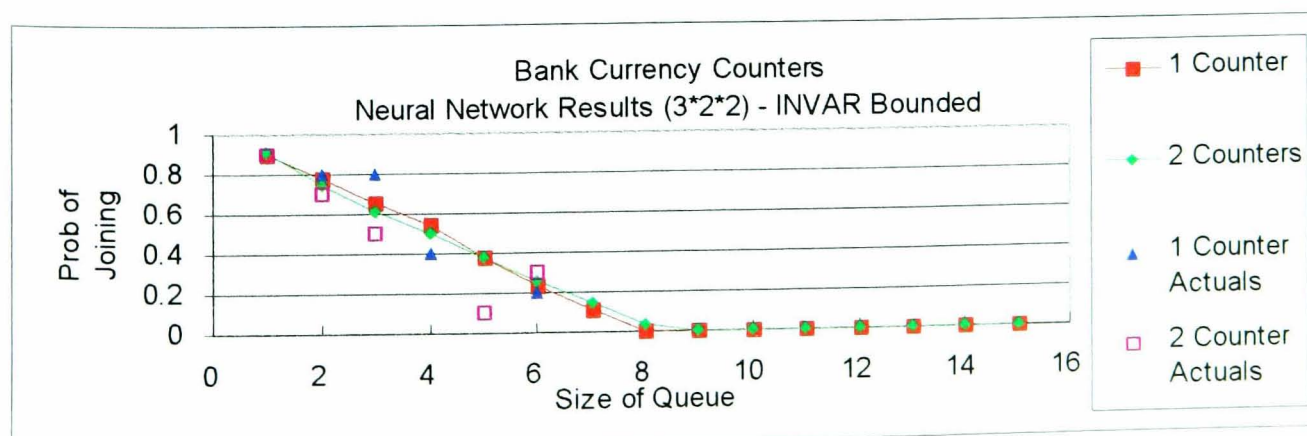
Number Counters	Shortest Length	Probability	Stay Decision	Balk Decision	Number Counters	Shortest Length	Probability	Stay Decision	Balk Decision
1	8	0.05	0	1	2	8	0.05	0	1
1	8	0.15	0	1	2	8	0.15	0	1
1	8	0.25	0	1	2	8	0.25	0	1
1	8	0.35	0	1	2	8	0.35	0	1
1	8	0.45	0	1	2	8	0.45	0	1
1	8	0.55	0	1	2	8	0.55	0	1
1	8	0.65	0	1	2	8	0.65	0	1
1	8	0.75	0	1	2	8	0.75	0	1
1	8	0.85	0	1	2	8	0.85	0	1
1	8	0.95	0	1	2	8	0.95	0	1

The results from the bounded INVAR models are shown below :

Counters Open	Queue Length	3*4*2		3*3*2		3*2*2		3*1*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	0.88	0.12	0.89	0.11	0.9	0.1	0.95	0.05		
1	2	0.83	0.17	0.84	0.16	0.78	0.22	0.8	0.2	0.8	0.2
1	3	0.75	0.25	0.76	0.24	0.66	0.34	0.65	0.35	0.8	0.2
1	4	0.44	0.56	0.44	0.56	0.54	0.46	0.5	0.5	0.4	0.6
1	5	0.24	0.76	0.27	0.73	0.38	0.62	0.35	0.65		
1	6	0.16	0.84	0.15	0.85	0.23	0.77	0.21	0.79	0.2	0.8
1	7	0.08	0.92	0.04	0.96	0.11	0.89	0.06	0.94		
1	8	0	1	0	1	0	1	0	1	0	1.0
1	9	0	1	0	1	0	1	0	1		
1	10	0	1	0	1	0	1	0	1		
1	11	0	1	0	1	0	1	0	1		
1	12	0	1	0	1	0	1	0	1		
1	13	0	1	0	1	0	1	0	1		
1	14	0	1	0	1	0	1	0	1		
1	15	0	1	0	1	0	1	0	1		
2	1	0.86	0.14	0.9	0.1	0.91	0.09	0.87	0.13	0.9	0.1
2	2	0.7	0.3	0.75	0.25	0.74	0.26	0.72	0.28	0.7	0.3
2	3	0.53	0.47	0.6	0.4	0.61	0.39	0.57	0.43	0.5	0.5
2	4	0.43	0.57	0.48	0.52	0.5	0.5	0.42	0.58		
2	5	0.33	0.67	0.37	0.63	0.38	0.62	0.28	0.72	0.1	0.9
2	6	0.23	0.77	0.25	0.75	0.26	0.74	0.13	0.87	0.3	0.7
2	7	0.13	0.87	0.14	0.86	0.15	0.85	0	1		
2	8	0.05	0.95	0.03	0.97	0.03	0.97	0	1	0	1.0
2	9	0	1	0	1	0	1	0	1		
2	10	0	1	0	1	0	1	0	1		
2	11	0	1	0	1	0	1	0	1		
2	12	0	1	0	1	0	1	0	1		
2	13	0	1	0	1	0	1	0	1		
2	14	0	1	0	1	0	1	0	1		
2	15	0	1	0	1	0	1	0	1		

Table of Results for Bounded Currency INVAR Models

The chosen model has a 3*2*2 architecture :



OUTDATA Models

The training data for the OUTDATA models is :

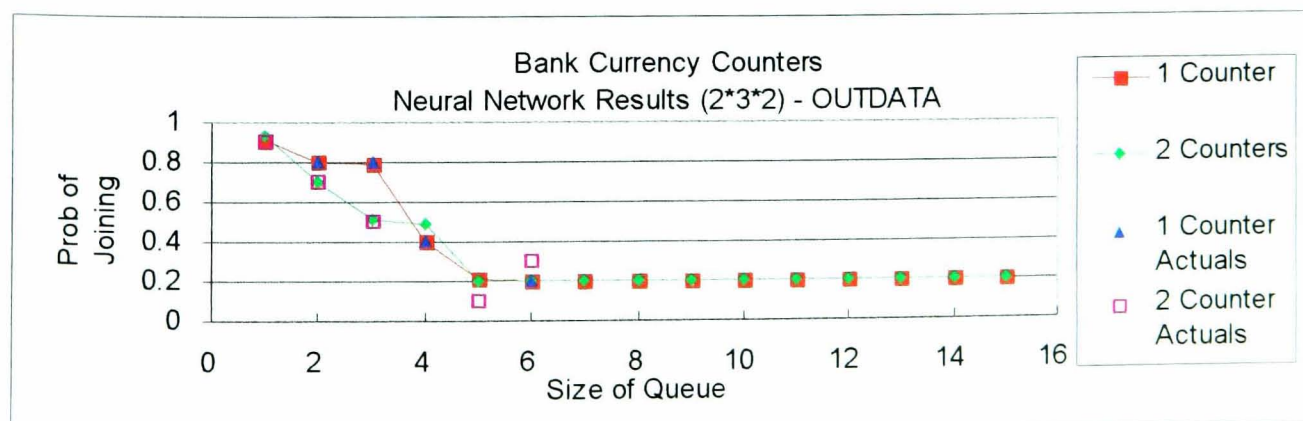
Number of Counters	Shortest Length	Probability of Staying	Probability of Leaving
1	2	0.8	0.2
1	3	0.8	0.2
1	4	0.4	0.6
1	6	0.2	0.8
2	1	0.9	0.1
2	2	0.7	0.3
2	3	0.5	0.5
2	4	0.5	0.5
2	5	0.1	0.9
2	6	0.3	0.7

The results from the OUTDATA models are :

Counters Open	Queue Length	2*6*2		2*4*2		2*3*2		2*2*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	0.53889	0.4613	0.84772	0.15228	0.9118	0.08829	0.92414	0.07586		
1	2	0.80012	0.19998	0.79991	0.2001	0.80273	0.19743	0.80588	0.19412	0.8	0.2
1	3	0.8002	0.19986	0.79995	0.20005	0.78456	0.21561	0.78977	0.21023	0.8	0.2
1	4	0.40031	0.59977	0.39999	0.60001	0.40111	0.59913	0.40771	0.59229	0.4	0.6
1	5	0.38497	0.61511	0.04159	0.95841	0.20835	0.79182	0.21152	0.78848		
1	6	0.20019	0.79992	0.19978	0.80022	0.20307	0.79709	0.20663	0.79337	0.2	0.8
1	7	0.26534	0.73501	0.71632	0.28368	0.20295	0.79721	0.20653	0.79347		
1	8	0.56231	0.4384	0.8867	0.1133	0.20295	0.79722	0.20653	0.79347		
1	9	0.83725	0.16322	0.92251	0.07749	0.20294	0.79722	0.20653	0.79347		
1	10	0.948	0.05216	0.93205	0.06795	0.20294	0.79722	0.20653	0.79347		
1	11	0.98135	0.01869	0.93492	0.06508	0.20294	0.79722	0.20653	0.79347		
1	12	0.99181	0.0082	0.93583	0.06417	0.20294	0.79722	0.20653	0.79347		
1	13	0.99559	0.00442	0.93612	0.06388	0.20294	0.79722	0.20653	0.79347		
1	14	0.99717	0.00283	0.93621	0.06379	0.20294	0.79722	0.20653	0.79347		
1	15	0.99792	0.00208	0.93624	0.06376	0.20294	0.79722	0.20653	0.79347		
2	1	0.90007	0.10004	0.90023	0.09977	0.93322	0.06684	0.93454	0.06546	0.9	0.1
2	2	0.70019	0.29999	0.69991	0.30009	0.69634	0.30388	0.70151	0.29849	0.7	0.3
2	3	0.50013	0.50006	0.50026	0.49974	0.5078	0.49245	0.50911	0.49089	0.5	0.5
2	4	0.50021	0.50007	0.49965	0.50035	0.48368	0.51657	0.48978	0.51022		
2	5	0.10016	0.89996	0.10002	0.89998	0.20546	0.7947	0.20817	0.79183	0.1	0.9
2	6	0.3003	0.6998	0.29975	0.70025	0.20295	0.79721	0.20653	0.79347	0.3	0.7
2	7	0.90651	0.09343	0.77658	0.22342	0.20294	0.79722	0.20653	0.79347		
2	8	0.98798	0.01201	0.8992	0.1008	0.20294	0.79722	0.20653	0.79347		
2	9	0.99602	0.00397	0.92578	0.07422	0.20294	0.79722	0.20653	0.79347		
2	10	0.99771	0.00229	0.93305	0.06695	0.20294	0.79722	0.20653	0.79347		
2	11	0.99829	0.00171	0.93525	0.06475	0.20294	0.79722	0.20653	0.79347		
2	12	0.99854	0.00145	0.93594	0.06406	0.20294	0.79722	0.20653	0.79347		
2	13	0.99868	0.00132	0.93615	0.06385	0.20294	0.79722	0.20653	0.79347		
2	14	0.99876	0.00124	0.93622	0.06378	0.20294	0.79722	0.20653	0.79347		
2	15	0.9988	0.0012	0.93625	0.06375	0.20294	0.79722	0.20653	0.79347		

Table of Results from Currency OUTDATA Models

The chosen model has a 2*3*2 architecture :



Bounded OUTDATA Models

Artificial data was added to the OUTDATA training data as shown below :

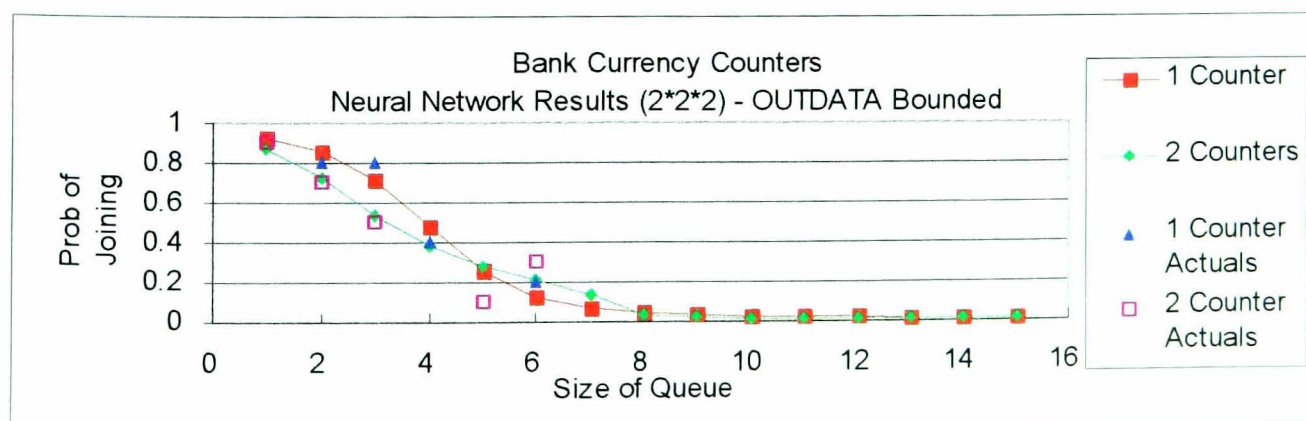
Number of Counters	Shortest Length	Probability of Staying	Probability of Leaving
1	8	1	0
2	8	1	0

The results from the Bounded OUTDATA models are :

Counters Open	Queue Length	2*4*2		2*3*2		2*2*2		2*1*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	0.7605	0.2395	0.90204	0.09796	0.9241	0.0759	0.91488	0.08512		
1	2	0.8048	0.1952	0.80439	0.19561	0.85274	0.14726	0.82683	0.17317	0.8	0.2
1	3	0.78276	0.21724	0.78488	0.21512	0.70663	0.29337	0.67928	0.32072	0.8	0.2
1	4	0.40212	0.59788	0.40323	0.59677	0.4746	0.5254	0.48998	0.51002	0.4	0.6
1	5	0.23684	0.76316	0.21103	0.78897	0.25323	0.74677	0.31278	0.68722		
1	6	0.19728	0.80272	0.20027	0.79973	0.12411	0.87589	0.18631	0.81369	0.2	0.8
1	7	0.00273	0.99727	0.08519	0.91481	0.06567	0.93433	0.11003	0.88997		
1	8	0.00007	0.99993	0.00006	0.99994	0.04019	0.95981	0.06718	0.93282	0	1.0
1	9	0.00007	0.99993	0.00001	0.99999	0.0284	0.9716	0.0433	0.9567		
1	10	0.00012	0.99988	0.00001	0.99999	0.0225	0.9775	0.02965	0.97035		
1	11	0.00016	0.99984	0.00001	0.99999	0.01935	0.98065	0.02156	0.97844		
1	12	0.0002	0.9998	0.00001	0.99999	0.01758	0.98242	0.01655	0.98345		
1	13	0.00023	0.99977	0.00001	0.99999	0.01656	0.98344	0.01333	0.98667		
1	14	0.00024	0.99976	0.00001	0.99999	0.01595	0.98405	0.01119	0.98881		
1	15	0.00025	0.99975	0.00001	0.99999	0.01559	0.98441	0.00971	0.99029		
2	1	0.97558	0.02442	0.93318	0.06682	0.87038	0.12962	0.86776	0.13224	0.9	0.1
2	2	0.68584	0.31416	0.69851	0.30149	0.72105	0.27895	0.74386	0.25614	0.7	0.3
2	3	0.50379	0.49621	0.51173	0.48827	0.53231	0.46769	0.56566	0.43434	0.5	0.5
2	4	0.49075	0.50925	0.48326	0.51674	0.37515	0.62485	0.37704	0.62296		
2	5	0.11251	0.88749	0.20878	0.79122	0.27409	0.72591	0.22896	0.77104	0.1	0.9
2	6	0.29034	0.70966	0.20457	0.79543	0.21084	0.78916	0.13483	0.86517	0.3	0.7
2	7	0.38495	0.61505	0.17893	0.82107	0.1284	0.8716	0.08098	0.91902		
2	8	0.00696	0.99304	0.00556	0.99444	0.03131	0.96869	0.05103	0.94897	0	1.0
2	9	0.00013	0.99987	0.00001	0.99999	0.01793	0.98207	0.03412	0.96588		
2	10	0.0001	0.9999	0.00001	0.99999	0.01637	0.98363	0.02424	0.97576		
2	11	0.00014	0.99986	0.00001	0.99999	0.01581	0.98419	0.01823	0.98177		
2	12	0.00018	0.99982	0.00001	0.99999	0.0155	0.9845	0.01443	0.98557		
2	13	0.00021	0.99979	0.00001	0.99999	0.01532	0.98468	0.01192	0.98808		
2	14	0.00023	0.99977	0.00001	0.99999	0.0152	0.9848	0.01023	0.98977		
2	15	0.00024	0.99976	0.00001	0.99999	0.01513	0.98487	0.00904	0.99096		

Table of Results from Bounded Currency OUTDATA Models

The selected model has a 2*2*2 architecture :



OUTNET Models

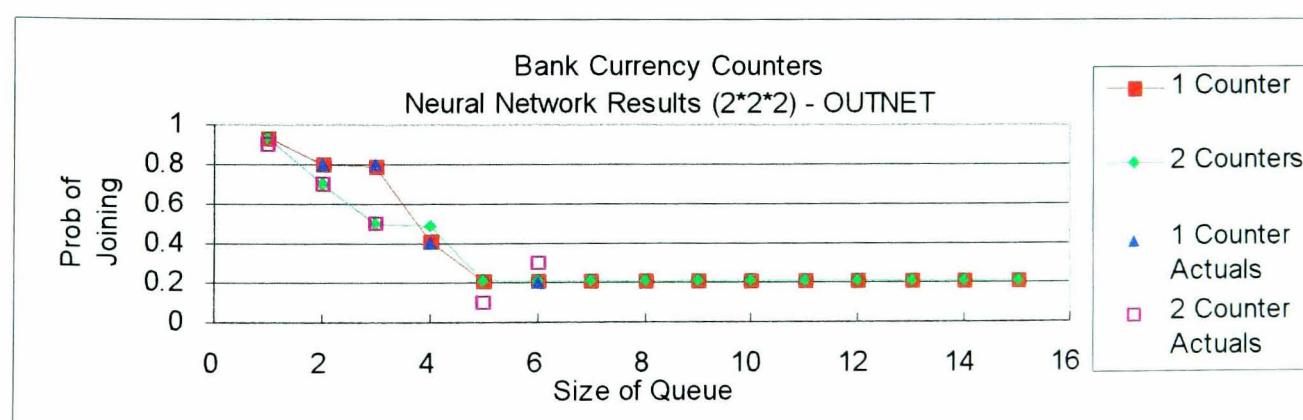
The training data for the OUTNET models is the same as for the INVAR models but with no probability input variable (see page H-28).

The results from the OUTNET models are :

Counters Open	Queue Length	2*4*2		2*3*2		2*2*2		2*1*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	0.85951	0.14049	0.87184	0.12816	0.93325	0.06485	0.88728	0.11272		
1	2	0.80482	0.19518	0.81434	0.18566	0.79932	0.19607	0.85422	0.14578	0.8	0.2
1	3	0.79424	0.20576	0.77834	0.22166	0.78514	0.21009	0.74564	0.25436	0.8	0.2
1	4	0.41471	0.58529	0.40563	0.59437	0.40734	0.5883	0.51935	0.48065	0.4	0.6
1	5	0.22111	0.77889	0.127	0.873	0.21182	0.78615	0.33694	0.66306		
1	6	0.21797	0.78203	0.20193	0.79807	0.20759	0.79043	0.26592	0.73408	0.2	0.8
1	7	0.21883	0.78117	0.48883	0.51117	0.20751	0.79051	0.24388	0.75612		
1	8	0.23619	0.76381	0.50352	0.49648	0.20751	0.79051	0.23729	0.76271		
1	9	0.50197	0.49803	0.50375	0.49625	0.20751	0.79051	0.23534	0.76466		
1	10	0.85745	0.14255	0.50375	0.49625	0.20751	0.79051	0.23476	0.76524		
1	11	0.88393	0.11607	0.50375	0.49625	0.20751	0.79051	0.23459	0.76541		
1	12	0.88527	0.11473	0.50375	0.49625	0.20751	0.79051	0.23453	0.76547		
1	13	0.88534	0.11466	0.50375	0.49625	0.20751	0.79051	0.23452	0.76548		
1	14	0.88534	0.11466	0.50375	0.49625	0.20751	0.79051	0.23452	0.76548		
1	15	0.88534	0.11466	0.50375	0.49625	0.20751	0.79051	0.23451	0.76549		
2	1	0.93175	0.06825	0.95076	0.04924	0.93166	0.06643	0.87088	0.12912	0.9	0.1
2	2	0.70843	0.29157	0.69083	0.30917	0.69774	0.29822	0.79835	0.20165	0.7	0.3
2	3	0.50509	0.49491	0.51564	0.48436	0.50405	0.49242	0.60714	0.39286	0.5	0.5
2	4	0.50827	0.49173	0.49034	0.50966	0.4891	0.50742	0.38684	0.61316		
2	5	0.10121	0.89879	0.11555	0.88445	0.2087	0.78931	0.2827	0.7173	0.1	0.9
2	6	0.30132	0.69868	0.30298	0.69702	0.20751	0.79051	0.24895	0.75105	0.3	0.7
2	7	0.35386	0.64614	0.49832	0.50168	0.20751	0.79051	0.2388	0.7612		
2	8	0.36897	0.63103	0.50366	0.49634	0.20751	0.79051	0.23579	0.76421		
2	9	0.38921	0.61079	0.50375	0.49625	0.20751	0.79051	0.23489	0.76511		
2	10	0.42061	0.57939	0.50375	0.49625	0.20751	0.79051	0.23462	0.76538		
2	11	0.4673	0.5327	0.50375	0.49625	0.20751	0.79051	0.23455	0.76545		
2	12	0.53117	0.46883	0.50375	0.49625	0.20751	0.79051	0.23452	0.76548		
2	13	0.60806	0.39194	0.50375	0.49625	0.20751	0.79051	0.23452	0.76548		
2	14	0.68635	0.31365	0.50375	0.49625	0.20751	0.79051	0.23451	0.76549		
2	15	0.75301	0.24699	0.50375	0.49625	0.20751	0.79051	0.23451	0.76549		

Table of Results from Currency OUTNET Models

The chosen network has a 2*2*2 architecture :



Bounded OUTNET Models

Artificial data was added to the OUTNET training set as follows :

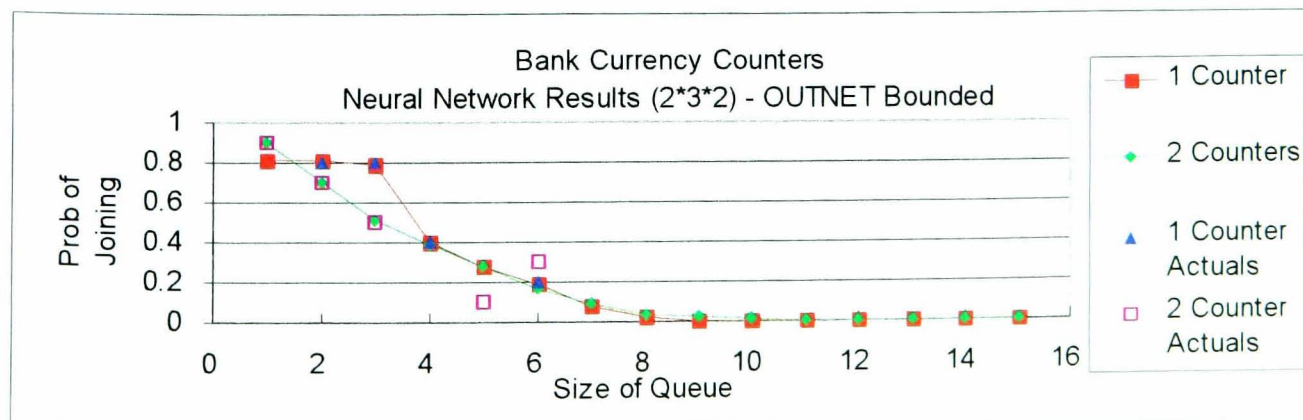
Number Counters	Shortest Length	Stay Decision	Balk Decision	Number Counters	Shortest Length	Stay Decision	Balk Decision
1	8	0	1	2	8	0	1
1	8	0	1	2	8	0	1
1	8	0	1	2	8	0	1
1	8	0	1	2	8	0	1
1	8	0	1	2	8	0	1
1	8	0	1	2	8	0	1
1	8	0	1	2	8	0	1
1	8	0	1	2	8	0	1
1	8	0	1	2	8	0	1
1	8	0	1	2	8	0	1

The results for the Bounded OUTNET models are :

Counters Open	Queue Length	2*4*2		2*3*2		2*2*2		2*1*2		Training	
		Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Prop. Join	Prop. Leave	Actual Join	Actual Leave
1	1	0.84851	0.15149	0.8096	0.1904	0.88249	0.11751	0.8918	0.1082		
1	2	0.84795	0.15205	0.80803	0.19197	0.81324	0.18676	0.81613	0.18387	0.8	0.2
1	3	0.7956	0.2044	0.78879	0.21121	0.77308	0.22692	0.67756	0.32244	0.8	0.2
1	4	0.40171	0.59829	0.39713	0.60287	0.40669	0.59331	0.48191	0.51809	0.4	0.6
1	5	0.36446	0.63554	0.28157	0.71843	0.14306	0.85694	0.29595	0.70405		
1	6	0.19887	0.80113	0.19192	0.80808	0.12166	0.87834	0.17353	0.82647	0.2	0.8
1	7	0.00005	0.99995	0.07757	0.92243	0.12016	0.87984	0.10794	0.89206		
1	8	0	1	0.01882	0.98118	0.12005	0.87995	0.07467	0.92533	0	1.0
1	9	0	1	0.00539	0.99461	0.12005	0.87995	0.05741	0.94259		
1	10	0	1	0.00271	0.99729	0.12005	0.87995	0.04806	0.95194		
1	11	0	1	0.00203	0.99797	0.12005	0.87995	0.04278	0.95722		
1	12	0	1	0.00183	0.99817	0.12005	0.87995	0.03971	0.96029		
1	13	0.00001	0.99999	0.00176	0.99824	0.12005	0.87995	0.03788	0.96212		
1	14	0.00001	0.99999	0.00174	0.99826	0.12005	0.87995	0.03678	0.96322		
1	15	0.00001	0.99999	0.00173	0.99827	0.12005	0.87995	0.0361	0.9639		
2	1	0.85142	0.14858	0.89836	0.10164	0.94335	0.05665	0.85306	0.14694	0.9	0.1
2	2	0.69972	0.30028	0.69722	0.30278	0.69069	0.30931	0.74235	0.25765	0.7	0.3
2	3	0.49675	0.50325	0.5106	0.4894	0.51509	0.48491	0.56482	0.43518	0.5	0.5
2	4	0.50142	0.49858	0.38361	0.61639	0.48621	0.51379	0.3652	0.6348		
2	5	0.10787	0.89213	0.27505	0.72495	0.12614	0.87386	0.21498	0.78502	0.1	0.9
2	6	0.29627	0.70373	0.17077	0.82923	0.12005	0.87995	0.12935	0.87065	0.3	0.7
2	7	0.30453	0.69547	0.08738	0.91262	0.12005	0.87995	0.08554	0.91446		
2	8	0.00442	0.99558	0.03857	0.96143	0.12005	0.87995	0.06313	0.93687	0	1.0
2	9	0.00003	0.99997	0.0167	0.9833	0.12005	0.87995	0.05121	0.94879		
2	10	0.00001	0.99999	0.00807	0.99193	0.12005	0.87995	0.04458	0.95542		
2	11	0.00001	0.99999	0.00464	0.99536	0.12005	0.87995	0.04077	0.95923		
2	12	0.00001	0.99999	0.00318	0.99682	0.12005	0.87995	0.03851	0.96149		
2	13	0.00001	0.99999	0.00249	0.99751	0.12005	0.87995	0.03716	0.96284		
2	14	0.00001	0.99999	0.00214	0.99786	0.12005	0.87995	0.03634	0.96366		
2	15	0.00001	0.99999	0.00196	0.99804	0.12005	0.87995	0.03583	0.96417		

Table of Results for Bounded Currency OUTNET Models

The chosen network has a 2*3*2 architecture :



Appendix I

Reneging Decision for the Bank Simulation

The unscaled training data for the renege decision is shown below :

Position Class	Worst Av. Service Time	Cum. Prob Reneging	Prob. Staying	Position Class	Worst Av. Service Time	Cum. Prob Reneging	Prob. Staying
0	0.04	0	1	0	1.01	0	1
0	0.04	0	1	0	1.01	0.0185	0.9815
0	0.04	0	1	0	1.1	0.0185	0.9815
0	0.04	0	1	0	1.1	0.0185	0.9815
0	0.04	0	1	0	1.14	0.037	0.963
0	0.04	0	1	0	1.2	0.037	0.963
0	0.04	0	1	0	1.2	0.037	0.963
0	0.04	0	1	0	1.2	0.037	0.963
0	0.04	0	1	0	1.2	0.037	0.963
0	0.04	0	1	0	1.2	0.037	0.963
0	0.38	0	1	0	1.21	0.037	0.963
0	0.38	0	1	0	1.21	0.037	0.963
0	0.38	0	1	0	1.21	0.037	0.963
0	0.38	0	1	0	1.21	0.037	0.963
0	0.38	0	1	0	1.21	0.037	0.963
0	0.38	0	1	0	1.21	0.037	0.963
0	0.38	0	1	0	1.21	0.037	0.963
0	0.38	0	1	0	1.21	0.037	0.963
0	0.38	0	1	0	1.21	0.037	0.963
0	0.44	0	1	0	1.22	0.037	0.963
0	0.44	0	1	0	1.22	0.037	0.963
0	0.44	0	1	0	1.22	0.037	0.963
0	0.44	0	1	0	1.22	0.037	0.963
0	0.56	0	1	0	1.22	0.037	0.963
0	0.56	0	1	0	1.22	0.037	0.963
0	0.56	0	1	0	1.22	0.037	0.963
0	0.56	0	1	0	1.22	0.037	0.963
0	0.56	0	1	0	1.22	0.037	0.963
0	0.56	0	1	0	1.25	0	1
0	0.56	0	1	0	1.25	0	1
0	0.57	0	1	0	1.25	0	1
0	0.62	0	1	0	1.25	0	1
0	0.62	0	1	0	1.25	0	1
0	0.62	0	1	0	1.25	0	1
0	0.66	0	1	0	1.25	0	1
0	0.66	0	1	0	1.26	0	1
0	0.66	0	1	0	1.26	0	1
0	0.66	0	1	0	1.26	0	1
0	0.66	0	1	0	1.38	0	1
0	0.66	0	1	0	1.49	0	1
0	0.66	0	1	0	1.49	0	1
0	0.66	0	1	0	1.6	0	1
0	0.66	0	1	0	1.73	0	1
0	0.66	0	1	0	1.73	0	1
0	0.77	0	1	0	1.73	0	1
0	0.77	0	1	0	1.73	0	1
0	0.77	0	1	0	1.73	0	1
0	0.8	0	1	0	1.73	0	1
0	0.8	0	1	0	1.74	0	1
0	0.8	0	1	0	1.74	0	1
0	0.8	0	1	0	1.74	0	1
0	0.8	0	1	0	1.74	0	1
0	0.8	0	1	0	1.74	0	1
0	0.81	0	1	0	1.74	0	1
0	0.84	0	1	0	1.74	0	1
0	0.84	0	1	0	1.74	0	1
0	0.84	0	1	0	1.74	0	1
0	0.86	0	1	0	1.83	0	1
0	0.99	0	1	0	1.86	0	1
0	0.99	0	1	0	1.86	0	1
0	0.99	0	1	0	1.87	0	1
0	0.99	0	1	0	1.87	0	1
0	0.99	0	1	0	1.87	0	1
0	0.99	0	1	0	1.91	0	1
0	0.99	0	1	0	1.91	0	1
0	0.99	0	1	0	1.94	0	1
0	0.99	0	1	0	1.94	0	1
0	0.99	0	1	0	1.94	0	1
0	1	0	1	0	1.94	0	1

0	1.94	0	1	0	3.74	0	1
0	1.95	0	1	0	3.74	0	1
0	1.95	0	1	0	3.74	0	1
0	1.95	0	1	0	3.74	0	1
0	1.95	0	1	0	3.74	0	1
0	1.96	0	1	0	3.74	0	1
0	1.96	0	1	0	3.74	0	1
0	1.96	0	1	0	3.96	0	1
0	1.96	0	1	0	3.96	0	1
0	1.96	0	1	0	3.96	0	1
0	2.04	0	1	0	3.96	0	1
0	2.04	0	1	0	3.96	0	1
0	2.04	0	1	0	3.96	0	1
0	2.06	0	1	0	4.08	0	1
0	2.12	0	1	0	4.08	0	1
0	2.15	0	1	0	4.08	0	1
0	2.15	0	1	0	4.08	0	1
0	2.17	0	1	0	4.35	0	1
0	2.17	0	1	0	4.4	0	1
0	2.17	0	1	0	4.4	0	1
0	2.17	0	1	0	4.4	0	1
0	2.17	0	1	0	4.4	0	1
0	2.17	0	1	0	4.68	0	1
0	2.17	0	1	0	4.68	0	1
0	2.24	0	1	0	5.21	0	1
0	2.24	0	1	0	5.55	0.025	0.975
0	2.24	0	1	0	5.59	0.025	0.975
0	2.24	0	1	0	5.59	0.025	0.975
0	2.24	0	1	0	5.59	0.025	0.975
0	2.24	0	1	0	5.59	0.025	0.975
0	2.24	0	1	0	5.59	0.025	0.975
0	2.24	0	1	0	5.59	0.025	0.975
0	2.24	0	1	0	6.22	0.025	0.975
0	2.24	0	1	0	6.26	0.025	0.975
0	2.24	0	1	0	6.26	0.025	0.975
0	2.24	0	1	0	6.26	0.025	0.975
0	2.24	0	1	0	6.26	0.025	0.975
0	2.3	0	1	0	6.26	0.025	0.975
0	2.3	0	1	0	6.26	0.025	0.975
0	2.36	0	1	0	6.26	0.025	0.975
0	2.42	0	1	0	6.26	0.025	0.975
0	2.44	0	1	0	6.26	0.025	0.975
0	2.44	0	1	0	6.26	0.025	0.975
0	2.44	0	1	0	6.26	0.025	0.975
0	2.44	0	1	0	6.26	0.025	0.975
0	2.44	0	1	0	7.15	0.025	0.975
0	2.44	0	1	0	7.15	0.025	0.975
0	2.44	0	1	0	7.15	0.025	0.975
0	2.44	0	1	0	7.15	0.025	0.975
0	2.44	0	1	0	7.15	0.025	0.975
0	2.44	0	1	0	7.15	0.025	0.975
0	2.44	0	1	0	7.15	0.025	0.975
0	2.44	0	1	0	7.89	0.025	0.975
0	2.6	0	1	0	7.89	0.025	0.975
0	2.6	0	1	0	7.89	0.025	0.975
0	2.6	0	1	0	7.89	0.025	0.975
0	2.6	0	1	0	7.89	0.025	0.975
0	2.6	0	1	0	7.89	0.025	0.975
0	2.6	0	1	0	7.97	0.025	0.975
0	2.6	0	1	0	8.53	0.025	0.975
0	2.79	0	1	0	8.53	0.025	0.975
0	2.79	0	1	0	8.53	0.025	0.975
0	2.79	0	1	0	8.53	0.025	0.975
0	2.79	0	1	0	8.53	0.025	0.975
0	2.79	0	1	0	8.53	0.025	0.975
0	2.79	0	1	0	8.53	0.025	0.975
0	2.79	0	1	0	11.4	0.05	0.95
0	2.79	0	1	1	0.25	0.0169	0.9831
0	2.8	0	1	1	1	0.0338	0.9662
0	2.8	0	1	1	1.1	0.0508	0.9492
0	2.8	0	1	1	1.1	0.0678	0.9322
0	2.85	0	1	1	1.5	0.0847	0.9153
0	2.89	0	1	1	1.92	0.0847	0.9153
0	3.15	0	1	1	2.07	0.1017	0.8983
0	3.15	0	1	1	2.11	0.1017	0.8983
0	3.15	0	1	1	2.4	0.1017	0.8983
0	3.15	0	1	1	2.45	0.1186	0.8814
0	3.15	0	1	1	2.49	0.1186	0.8814
0	3.15	0	1	1	2.49	0.1186	0.8814
0	3.15	0	1	1	2.49	0.1186	0.8814
0	3.15	0	1	1	2.49	0.1186	0.8814
0	3.15	0	1	1	2.49	0.1186	0.8814
0	3.19	0	1	1	2.49	0.1186	0.8814
0	3.3	0	1	1	2.61	0.1186	0.8814
0	3.34	0	1	1	2.63	0.1186	0.8814
0	3.45	0	1	1	2.63	0.1186	0.8814
0	3.6	0	1	1	2.63	0.1186	0.8814
0	3.6	0	1	1	2.63	0.1186	0.8814
0	3.6	0	1	1	2.63	0.1186	0.8814

1	2.76	0.1186	0.8814	1	3.84	0.1356	0.8644
1	2.76	0.1186	0.8814	1	3.87	0.1356	0.8644
1	2.81	0.1186	0.8814	1	3.87	0.1356	0.8644
1	2.81	0.1186	0.8814	1	4.21	0.1356	0.8644
1	2.81	0.1186	0.8814	1	4.21	0.1356	0.8644
1	2.87	0.1356	0.8644	1	4.4	0.1525	0.8475
1	2.96	0.1356	0.8644	1	4.42	0.1695	0.8305
1	3.11	0.1356	0.8644	1	4.53	0.1695	0.8305
1	3.15	0.1356	0.8644	1	4.53	0.1695	0.8305
1	3.26	0.1356	0.8644	1	4.53	0.1695	0.8305
1	3.26	0.1356	0.8644	1	4.53	0.1695	0.8305
1	3.28	0.1356	0.8644	1	4.53	0.1695	0.8305
1	3.29	0.1356	0.8644	1	4.53	0.1695	0.8305
1	3.29	0.1356	0.8644	1	5.01	0.1864	0.8136
1	3.29	0.1356	0.8644	1	5.01	0.1864	0.8136
1	3.39	0.1356	0.8644	1	5.01	0.1864	0.8136
1	3.39	0.1356	0.8644	1	5.01	0.1864	0.8136
1	3.39	0.1356	0.8644	1	5.03	0.1864	0.8136
1	3.56	0.1356	0.8644	1	5.2	0.1864	0.8136

Note that the Position Class has values of :

0 : if the customer is 3rd from the front or closer

1 : if the customer is 4th from the front or further

where in the queue is the next person to be served (i.e. the person who is being served is excluded from the figure).

The results from training networks with different numbers of hidden nodes are shown below :

Queue Pos	Av. Serve Time	2*4*2		2*6*2		2*8*2		Queue Pos	Av. Serve Time	2*4*2		2*6*2		2*8*2	
		Renege	Stay	Renege	Stay	Renege	Stay			Renege	Stay	Renege	Stay	Renege	Stay
0	0.027147	0.00656	0.99344	0.00672	0.99329	0.00526	0.99473	0	1.263028	0.00539	0.99461	0.0059	0.9941	0.00724	0.99276
0	0.056901	0.00655	0.99345	0.00672	0.99328	0.00535	0.99463	0	1.275901	0.00537	0.99463	0.00588	0.99412	0.00722	0.99278
0	0.130982	0.00653	0.99347	0.00673	0.99327	0.00559	0.9944	0	1.307435	0.00532	0.99468	0.00582	0.99418	0.00717	0.99284
0	0.141994	0.00652	0.99348	0.00673	0.99327	0.00562	0.99437	0	1.307879	0.00532	0.99468	0.00582	0.99418	0.00717	0.99284
0	0.178529	0.00651	0.99349	0.00674	0.99327	0.00573	0.99425	0	1.343751	0.00526	0.99474	0.00575	0.99425	0.0071	0.9929
0	0.206675	0.00649	0.99351	0.00674	0.99327	0.00582	0.99417	0	1.351983	0.00525	0.99475	0.00574	0.99426	0.00708	0.99292
0	0.285404	0.00645	0.99355	0.00674	0.99327	0.00606	0.99393	0	1.364246	0.00523	0.99477	0.00571	0.99429	0.00705	0.99295
0	0.287615	0.00645	0.99355	0.00674	0.99327	0.00607	0.99392	0	1.373051	0.00521	0.99479	0.0057	0.9943	0.00704	0.99297
0	0.289316	0.00645	0.99355	0.00674	0.99327	0.00607	0.99392	0	1.383916	0.00519	0.99481	0.00568	0.99432	0.00701	0.99299
0	0.338943	0.00642	0.99358	0.00673	0.99327	0.00622	0.99377	0	1.417272	0.00514	0.99486	0.00561	0.99439	0.00693	0.99307
0	0.356339	0.00641	0.99359	0.00673	0.99328	0.00627	0.99372	0	1.453994	0.00507	0.99493	0.00553	0.99447	0.00684	0.99317
0	0.356646	0.00641	0.99359	0.00673	0.99328	0.00627	0.99372	0	1.458586	0.00506	0.99494	0.00552	0.99448	0.00682	0.99318
0	0.390191	0.00639	0.99361	0.00672	0.99328	0.00636	0.99363	0	1.476486	0.00503	0.99497	0.00548	0.99452	0.00677	0.99323
0	0.432269	0.00636	0.99364	0.00671	0.99329	0.00648	0.99351	0	1.501207	0.00499	0.99501	0.00543	0.99457	0.0067	0.9933
0	0.483152	0.00632	0.99368	0.0067	0.99331	0.00661	0.99338	0	1.519032	0.00496	0.99504	0.00539	0.99461	0.00665	0.99335
0	0.484337	0.00632	0.99368	0.0067	0.99331	0.00662	0.99338	0	1.54747	0.00491	0.99509	0.00533	0.99467	0.00656	0.99344
0	0.496337	0.00631	0.99369	0.00669	0.99331	0.00665	0.99334	0	1.59733	0.00481	0.99519	0.00521	0.99479	0.00639	0.99361
0	0.602451	0.00622	0.99378	0.00665	0.99336	0.0069	0.9931	0	1.634783	0.00474	0.99526	0.00512	0.99488	0.00626	0.99375
0	0.631395	0.0062	0.9938	0.00663	0.99337	0.00696	0.99303	0	1.650424	0.00471	0.99529	0.00508	0.99492	0.0062	0.9938
0	0.698691	0.00613	0.99387	0.00659	0.99342	0.00709	0.9929	0	1.657131	0.0047	0.9953	0.00507	0.99493	0.00618	0.99383
0	0.705583	0.00612	0.99388	0.00658	0.99342	0.0071	0.99289	0	1.679235	0.00466	0.99534	0.00501	0.99499	0.00609	0.99391
0	0.707724	0.00612	0.99388	0.00658	0.99342	0.00711	0.99289	0	1.708545	0.0046	0.9954	0.00494	0.99506	0.00598	0.99403
0	0.726168	0.0061	0.9939	0.00657	0.99344	0.00714	0.99286	0	1.719413	0.00458	0.99542	0.00491	0.99509	0.00593	0.99407
0	0.749423	0.00608	0.99392	0.00655	0.99345	0.00718	0.99282	0	1.728161	0.00457	0.99543	0.00489	0.99511	0.0059	0.99411
0	0.770108	0.00606	0.99394	0.00653	0.99347	0.00721	0.99279	0	1.784563	0.00446	0.99554	0.00475	0.99525	0.00566	0.99434
0	0.780587	0.00605	0.99395	0.00652	0.99348	0.00722	0.99277	0	1.790856	0.00444	0.99556	0.00473	0.99527	0.00564	0.99437
0	0.797426	0.00603	0.99397	0.00651	0.99349	0.00725	0.99275	0	1.810118	0.00441	0.99559	0.00468	0.99532	0.00555	0.99445
0	0.84464	0.00597	0.99403	0.00647	0.99354	0.0073	0.99269	0	1.887101	0.00426	0.99574	0.00447	0.99553	0.00522	0.99479
0	0.872083	0.00594	0.99406	0.00644	0.99356	0.00733	0.99267	0	1.92922	0.00417	0.99583	0.00436	0.99564	0.00503	0.99498
0	0.917867	0.00588	0.99412	0.00639	0.99361	0.00737	0.99263	0	2.022149	0.00399	0.99601	0.0041	0.9959	0.0046	0.99541
0	0.939862	0.00586	0.99414	0.00637	0.99363	0.00738	0.99262	0	2.04117	0.00395	0.99605	0.00405	0.99595	0.00451	0.99549
0	0.940845	0.00586	0.99414	0.00637	0.99363	0.00738	0.99262	0	2.043218	0.00394	0.99606	0.00404	0.99596	0.0045	0.9955
0	0.965433	0.00582	0.99418	0.00634	0.99366	0.00739	0.9926	0	2.109913	0.00381	0.99619	0.00385	0.99615	0.00419	0.99581
0	0.965985	0.00582	0.99418	0.00634	0.99366	0.00739	0.9926	0	2.11263	0.00381	0.99619	0.00385	0.99615	0.00418	0.99582
0	0.998843	0.00578	0.99422	0.0063	0.9937	0.0074	0.99259	0	2.135978	0.00376	0.99624	0.00378	0.99622	0.00407	0.99593
0	1.101567	0.00564	0.99436	0.00616	0.99384	0.00739	0.99261	0	2.178961	0.00367	0.99633	0.00366	0.99634	0.00388	0.99613
0	1.108813	0.00563	0.99437	0.00615	0.99385	0.00739	0.99261	0	2.190876	0.00365	0.99635	0.00362	0.99638	0.00382	0.99618
0	1.133107	0.00559	0.99441	0.00611	0.99389	0.00738	0.99262	0	2.211004	0.00361	0.99639	0.00357	0.99643	0.00373	0.99627
0	1.15314	0.00556	0.99444	0.00608	0.99392	0.00736	0.99264	0	2.216057	0.0036	0.9964	0.00355	0.99645	0.00371	0.99629
0	1.168842	0.00554	0.99446	0.00606	0.99394	0.00735	0.99265	0	2.217945	0.0036	0.9964	0.00355	0.99645	0.0037	0.9963
0	1.195781	0.0055	0.9945	0.00602	0.99399	0.00732	0.99268	0	2.234939	0.00356	0.99644	0.0035	0.9965	0.00363	0.99638

Queue Pos	Av. Serve Time	2*4*2		2*6*2		2*8*2		Queue Pos	Av. Serve Time	2*4*2		2*6*2		2*8*2	
		Reneg	Stay	Reneg	Stay	Reneg	Stay			Reneg	Stay	Reneg	Stay	Reneg	Stay
0	2.261987	0.00351	0.99649	0.00343	0.99658	0.00351	0.9965	0	5.466175	0.01833	0.98167	0.01898	0.98103	0.01854	0.98147
0	2.266553	0.0035	0.9965	0.00341	0.99659	0.00349	0.99652	0	5.480833	0.01859	0.98141	0.01925	0.98076	0.01878	0.98123
0	2.267885	0.0035	0.9965	0.00341	0.99659	0.00348	0.99652	0	5.529798	0.01944	0.98056	0.02011	0.97991	0.01953	0.98048
0	2.316006	0.00341	0.99659	0.00328	0.99673	0.00328	0.99673	0	5.549576	0.01977	0.98023	0.02044	0.97958	0.01982	0.98019
0	2.363529	0.00331	0.99669	0.00314	0.99686	0.00308	0.99692	0	5.563219	0.02	0.98	0.02066	0.97935	0.02002	0.98
0	2.444078	0.00316	0.99684	0.00293	0.99707	0.00278	0.99724	0	5.621012	0.02089	0.97911	0.02154	0.97847	0.02078	0.97923
0	2.467456	0.00312	0.99688	0.00287	0.99713	0.00287	0.99733	0	5.658533	0.02143	0.97857	0.02206	0.97795	0.02124	0.97878
0	2.499861	0.00306	0.99694	0.00278	0.99722	0.00255	0.99745	0	5.667156	0.02155	0.97845	0.02218	0.97784	0.02134	0.97868
0	2.551174	0.00296	0.99704	0.00285	0.99735	0.00237	0.99763	0	5.69294	0.0219	0.9781	0.02251	0.97751	0.02162	0.97839
0	2.553974	0.00296	0.99704	0.00264	0.99736	0.00236	0.99764	0	5.723644	0.02228	0.97772	0.02287	0.97714	0.02194	0.97807
0	2.565009	0.00294	0.99706	0.00262	0.99738	0.00232	0.99768	0	5.735999	0.02243	0.97757	0.02301	0.977	0.02206	0.97795
0	2.607983	0.00286	0.99714	0.00251	0.99749	0.00218	0.99782	0	5.740123	0.02248	0.97752	0.02306	0.97695	0.0221	0.97791
0	2.609255	0.00286	0.99714	0.00251	0.99749	0.00217	0.99783	0	5.749178	0.02259	0.97741	0.02316	0.97685	0.02219	0.97783
0	2.623446	0.00284	0.99716	0.00247	0.99753	0.00213	0.99787	0	5.768426	0.02281	0.97719	0.02336	0.97665	0.02237	0.97765
0	2.641038	0.00281	0.99719	0.00243	0.99757	0.00207	0.99793	0	5.868273	0.0238	0.9762	0.02425	0.97576	0.02315	0.97686
0	2.708662	0.00269	0.99731	0.00227	0.99773	0.00187	0.99813	0	5.880613	0.02391	0.97609	0.02435	0.97567	0.02323	0.97678
0	2.733909	0.00265	0.99735	0.00222	0.99778	0.0018	0.9982	0	5.930038	0.0243	0.9757	0.02468	0.97533	0.02354	0.97648
0	2.767271	0.0026	0.9974	0.00214	0.99786	0.00172	0.99828	0	5.959284	0.0245	0.9755	0.02485	0.97516	0.02369	0.97632
0	2.776431	0.00258	0.99742	0.00212	0.99788	0.00169	0.99831	0	6.005006	0.02479	0.97521	0.02508	0.97493	0.02391	0.9761
0	2.824103	0.00251	0.99749	0.00202	0.99798	0.00158	0.99843	0	6.016031	0.02486	0.97514	0.02513	0.97489	0.02396	0.97605
0	2.846558	0.00247	0.99753	0.00198	0.99802	0.00152	0.99848	0	6.065439	0.02511	0.97489	0.02532	0.9747	0.02414	0.97586
0	2.919817	0.00237	0.99763	0.00184	0.99816	0.00137	0.99863	0	6.088303	0.02521	0.97479	0.02539	0.97462	0.02422	0.97579
0	2.924234	0.00236	0.99764	0.00183	0.99817	0.00136	0.99864	0	6.099722	0.02526	0.97474	0.02542	0.97459	0.02426	0.97575
0	2.940697	0.00234	0.99766	0.0018	0.9982	0.00133	0.99868	0	6.131051	0.02539	0.97461	0.0255	0.97451	0.02434	0.97566
0	3.047961	0.00219	0.99781	0.00162	0.99838	0.00114	0.99886	0	6.137725	0.02541	0.97459	0.02552	0.97449	0.02436	0.97564
0	3.055273	0.00218	0.99782	0.00161	0.99839	0.00113	0.99887	0	6.138277	0.02541	0.97459	0.02552	0.97449	0.02436	0.97564
0	3.084924	0.00215	0.99785	0.00156	0.99844	0.00108	0.99892	0	6.153402	0.02546	0.97454	0.02555	0.97446	0.0244	0.97561
0	3.089619	0.00214	0.99786	0.00156	0.99844	0.00108	0.99892	0	6.201818	0.02561	0.97439	0.02563	0.97438	0.0245	0.9755
0	3.109335	0.00212	0.99788	0.00153	0.99847	0.00105	0.99895	0	6.209697	0.02563	0.97437	0.02564	0.97437	0.02452	0.97549
0	3.115649	0.00211	0.99789	0.00152	0.99848	0.00104	0.99896	0	6.227534	0.02567	0.97433	0.02566	0.97435	0.02455	0.97545
0	3.135825	0.00209	0.99791	0.00149	0.99851	0.00101	0.99899	0	6.267347	0.02576	0.97424	0.0257	0.97432	0.02461	0.97539
0	3.138617	0.00209	0.99791	0.00149	0.99851	0.00101	0.99899	0	6.277337	0.02578	0.97422	0.0257	0.97431	0.02462	0.97538
0	3.149436	0.00207	0.99793	0.00147	0.99853	0.001	0.999	0	6.310377	0.02583	0.97417	0.02571	0.9743	0.02466	0.97534
0	3.280372	0.00194	0.99806	0.00132	0.99868	0.00086	0.99914	0	6.32315	0.02585	0.97415	0.02571	0.9743	0.02467	0.97533
0	3.30655	0.00192	0.99808	0.00129	0.99871	0.00084	0.99916	0	6.324205	0.02585	0.97415	0.02571	0.9743	0.02467	0.97533
0	3.330725	0.0019	0.9981	0.00127	0.99873	0.00082	0.99918	0	6.351545	0.02588	0.97412	0.02571	0.9743	0.0247	0.9753
0	3.339473	0.00189	0.99811	0.00126	0.99874	0.00081	0.99919	0	6.407444	0.02592	0.97408	0.0257	0.97431	0.02473	0.97527
0	3.365573	0.00187	0.99813	0.00124	0.99876	0.00079	0.99921	0	6.409232	0.02593	0.97407	0.0257	0.97432	0.02473	0.97527
0	3.373387	0.00186	0.99814	0.00123	0.99877	0.00079	0.99921	0	6.42384	0.02593	0.97407	0.02569	0.97432	0.02474	0.97526
0	3.413631	0.00184	0.99816	0.0012	0.9988	0.00076	0.99924	0	6.435803	0.02594	0.97406	0.02568	0.97433	0.02474	0.97526
0	3.491107	0.00179	0.99821	0.00115	0.99885	0.00073	0.99927	0	6.441707	0.02594	0.97406	0.02568	0.97433	0.02474	0.97525
0	3.503455	0.00178	0.99822	0.00114	0.99886	0.00072	0.99928	0	6.477036	0.02595	0.97405	0.02565	0.97436	0.02475	0.97524
0	3.512968	0.00178	0.99822	0.00114	0.99886	0.00072	0.99928	0	6.504467	0.02595	0.97405	0.02562	0.97439	0.02476	0.97524
0	3.528533	0.00177	0.99823	0.00113	0.99887	0.00072	0.99928	0	6.508111	0.02595	0.97405	0.02562	0.97439	0.02476	0.97524
0	3.553654	0.00176	0.99824	0.00112	0.99888	0.00071	0.99929	0	6.519567	0.02595	0.97405	0.02561	0.9744	0.02476	0.97524
0	3.651593	0.00174	0.99826	0.0011	0.99891	0.0007	0.9993	0	6.533427	0.02595	0.97405	0.0256	0.97442	0.02476	0.97524
0	3.664107	0.00173	0.99827	0.00109	0.99891	0.0007	0.9993	0	6.553571	0.02594	0.97406	0.02557	0.97444	0.02476	0.97523
0	3.671927	0.00173	0.99827	0.00109	0.99891	0.0007	0.9993	0	6.667446	0.02588	0.97412	0.02543	0.97458	0.02475	0.97524
0	3.699735	0.00173	0.99827	0.00109	0.99891	0.00071	0.99929	0	6.668816	0.02588	0.97412	0.02543	0.97458	0.02475	0.97524
0	3.702438	0.00173	0.99827	0.00109	0.99891	0.00071	0.99929	0	6.716955	0.02584	0.97416	0.02537	0.97464	0.02474	0.97525
0	3.702523	0.00173	0.99827	0.00109	0.99891	0.00071	0.99929	0	6.734382	0.02583	0.97417	0.02535	0.97466	0.02474	0.97525
0	3.708486	0.00173	0.99827	0.00109	0.99891	0.00071	0.99929	0	6.743498	0.02582	0.97418	0.02533	0.97468	0.02473	0.97525
0	3.744413	0.00174	0.99826	0.0011	0.9989	0.00073	0.99927	0	6.940733	0.02561	0.97439	0.02509	0.97492	0.0247	0.97528
0	3.793068	0.00174	0.99826	0.00111	0.99889	0.00074	0.99926	0	7.005285	0.02554	0.97446	0.02502	0.97499	0.0247	0.97528
0	3.801917	0.00175	0.99825	0.00111	0.99889	0.00074	0.99926	0	7.036926	0.02551	0.97449	0.02499	0.97502	0.0247	0.97528
0	3.812018	0.00175	0.99825	0.00111	0.99889	0.00074	0.99926	0	7.045579	0.0255	0.9745	0.02498	0.97503	0.02471	0.97528
0	3.834714	0.00176	0.99824	0.00112	0.99888	0.00076	0.99924	0	7.062064	0.02548	0.97452	0.02497	0.97504	0.02471	0.97527
0	3.843431	0.00176	0.99824	0.00113	0.99887	0.00076	0.99924	0	7.066166	0.02547	0.97453	0.02496	0.97505	0.02471	0.97527
0	3.866463	0.00177	0.99823	0.00114	0.99886	0.00078	0.99922	0	7.107838	0.02543	0.97457	0.02493	0.97508	0.02471	0.97526
0	3.926303	0.00181	0.99819	0.00118	0.99882	0.00083	0.99917	0	7.157349	0.02538	0.97462	0.0249	0.97511	0.02473	0.97525
0	3.972517	0.00185	0.99815	0.00122	0.99878	0.00088	0.99912	0	7.232877	0.0253	0.9747	0.02486	0.97515	0.02475	0.97522
0	4.002843	0.00188	0.99812	0.00126	0.99874	0.00091	0.99909	0	7.241468	0.02529	0.97471	0.02485	0.97515	0.02476	0.97522
0	4.121887	0.00206	0.99794	0.00144	0.99856	0.00111	0.99889	0	7.246016	0.02529	0.97471	0.02485	0.97515	0.02476	0.97522
0	4.134095	0.00208	0.99792	0.00147	0.99854	0.00114	0.99886	0	7.309123	0.02523	0.97477	0.02484	0.97517	0.02479	0.97518
0	4.137432	0.00208	0.99792	0.00147	0.99853	0.00115	0.99885	0	7.331719	0.02					

Queue Pos	Av. Serve Time	2'4"2		2'6"2		2'8"2		Queue Pos	Av. Serve Time	2'4"2		2'6"2		2'8"2	
		Reneg	Stay	Reneg	Stay	Reneg	Stay			Reneg	Stay	Reneg	Stay	Reneg	Stay
0	8.403732	0.02526	0.97474	0.02641	0.97359	0.02678	0.97316	0	10.86931	0.04002	0.95998	0.0425	0.95747	0.04124	0.95857
0	8.421518	0.02528	0.97472	0.02646	0.97354	0.02683	0.97311	0	10.87211	0.04007	0.95993	0.04254	0.95744	0.04127	0.95854
0	8.453541	0.02531	0.97469	0.02656	0.97344	0.02693	0.97301	0	10.87235	0.04007	0.95993	0.04254	0.95743	0.04127	0.95854
0	8.459938	0.02532	0.97468	0.02658	0.97342	0.02695	0.97299	0	10.88707	0.04029	0.95971	0.04271	0.95727	0.04142	0.95839
0	8.466948	0.02533	0.97467	0.0266	0.9734	0.02697	0.97297	0	10.90031	0.04049	0.95951	0.04286	0.95712	0.04155	0.95826
0	8.523261	0.0254	0.9746	0.02678	0.97322	0.02715	0.97279	0	10.91005	0.04063	0.95937	0.04297	0.957	0.04165	0.95816
0	8.561694	0.02546	0.97454	0.02691	0.97309	0.02727	0.97266	0	10.94243	0.04113	0.95887	0.04335	0.95663	0.04198	0.95783
0	8.575421	0.02548	0.97452	0.02695	0.97304	0.02732	0.97261	0	10.96227	0.04144	0.95856	0.04358	0.95639	0.04218	0.95762
0	8.592391	0.02551	0.97449	0.02701	0.97299	0.02738	0.97256	0	10.97096	0.04158	0.95842	0.04368	0.95629	0.04227	0.95753
0	8.644873	0.02559	0.97441	0.0272	0.9728	0.02755	0.97237	0	10.99249	0.04193	0.95807	0.04394	0.95603	0.04249	0.95731
0	8.6592	0.02561	0.97439	0.02725	0.97275	0.0276	0.97232	0	11.00742	0.04217	0.95783	0.04412	0.95585	0.04265	0.95715
0	8.702221	0.02569	0.97431	0.02741	0.97259	0.02776	0.97217	0	11.01565	0.0423	0.9577	0.04422	0.95575	0.04274	0.95706
0	8.729645	0.02574	0.97426	0.02751	0.97249	0.02786	0.97207	0	11.03714	0.04266	0.95734	0.04448	0.95549	0.04297	0.95683
0	8.735689	0.02575	0.97425	0.02753	0.97246	0.02788	0.97205	0	11.06845	0.04319	0.95681	0.04487	0.9551	0.0433	0.95649
0	8.765007	0.02581	0.97419	0.02765	0.97235	0.02799	0.97194	0	11.07398	0.04329	0.95671	0.04494	0.95503	0.04336	0.95643
0	8.798327	0.02588	0.97412	0.02778	0.97222	0.02811	0.97181	0	11.09074	0.04358	0.95642	0.04515	0.95482	0.04355	0.95624
0	8.811522	0.02591	0.97409	0.02783	0.97217	0.02816	0.97176	0	11.23308	0.04621	0.95379	0.04702	0.95295	0.04517	0.95461
0	8.824413	0.02593	0.97407	0.02788	0.97211	0.02821	0.97171	0	11.29491	0.04745	0.95255	0.04787	0.95209	0.04591	0.95386
0	8.845541	0.02598	0.97402	0.02797	0.97203	0.02829	0.97163	0	11.33395	0.04825	0.95175	0.04843	0.95154	0.04639	0.95338
0	8.9091	0.02612	0.97388	0.02823	0.97176	0.02854	0.97138	0	11.49692	0.05189	0.94811	0.05087	0.94909	0.04851	0.95125
0	8.912256	0.02613	0.97387	0.02825	0.97175	0.02855	0.97136	0	11.63374	0.05527	0.94473	0.05309	0.94687	0.05043	0.94931
0	8.919667	0.02615	0.97385	0.02828	0.97172	0.02858	0.97133	0	11.71109	0.05733	0.94267	0.05442	0.94554	0.05158	0.94815
0	9.00245	0.02636	0.97364	0.02864	0.97135	0.02892	0.97099	0	11.73759	0.05806	0.94194	0.05489	0.94507	0.05198	0.94774
0	9.011714	0.02638	0.97362	0.02868	0.97131	0.02896	0.97095	0	11.73856	0.05808	0.94192	0.05491	0.94505	0.052	0.94773
0	9.041367	0.02646	0.97354	0.02882	0.97118	0.02909	0.97083	0	11.80582	0.05999	0.94001	0.05613	0.94383	0.05306	0.94666
0	9.04879	0.02649	0.97351	0.02885	0.97114	0.02912	0.97079	0	11.80804	0.06005	0.93995	0.05617	0.94379	0.05309	0.94663
0	9.061517	0.02652	0.97348	0.02891	0.97108	0.02917	0.97074	0	11.81676	0.0603	0.9397	0.05633	0.94362	0.05323	0.94649
0	9.098122	0.02663	0.97337	0.02908	0.97091	0.02933	0.97058	0	11.82741	0.06062	0.93938	0.05653	0.94342	0.0534	0.94631
0	9.105872	0.02665	0.97335	0.02912	0.97087	0.02937	0.97054	0	11.93219	0.06378	0.93622	0.05855	0.9414	0.05515	0.94455
0	9.108722	0.02666	0.97334	0.02913	0.97086	0.02938	0.97053	0	11.95092	0.06437	0.93563	0.05892	0.94103	0.05547	0.94423
0	9.155312	0.0268	0.9732	0.02936	0.97063	0.02959	0.97032	0	12.01816	0.06653	0.93347	0.0603	0.93965	0.05666	0.94303
0	9.209038	0.02698	0.97303	0.02963	0.97037	0.02983	0.97007	0	12.0188	0.06655	0.93345	0.06031	0.93964	0.05667	0.94302
0	9.233758	0.02706	0.97294	0.02975	0.97024	0.02995	0.96996	0	12.07031	0.06826	0.93174	0.0614	0.93855	0.05761	0.94207
0	9.240455	0.02708	0.97292	0.02979	0.97021	0.02998	0.96992	0	12.07457	0.06841	0.93159	0.06149	0.93846	0.05769	0.94199
0	9.242356	0.02709	0.97291	0.0298	0.9702	0.02999	0.96992	0	12.08091	0.06862	0.93138	0.06163	0.93832	0.05781	0.94187
0	9.267445	0.02718	0.97282	0.02993	0.97007	0.03011	0.9698	0	12.08318	0.0687	0.9313	0.06167	0.93827	0.05785	0.94183
0	9.2863	0.02724	0.97276	0.03002	0.96997	0.0302	0.96971	0	12.10055	0.06929	0.93071	0.06205	0.93789	0.05818	0.9415
0	9.298165	0.02729	0.97271	0.03009	0.96991	0.03025	0.96965	0	12.14358	0.07078	0.92922	0.063	0.93694	0.059	0.94067
0	9.318405	0.02736	0.97264	0.03019	0.9698	0.03035	0.96955	0	12.1464	0.07088	0.92912	0.06306	0.93688	0.05905	0.94062
0	9.365404	0.02754	0.97246	0.03045	0.96954	0.03058	0.96932	0	12.14692	0.07089	0.92911	0.06308	0.93687	0.05906	0.94061
0	9.437901	0.02783	0.97217	0.03085	0.96914	0.03095	0.96895	0	12.15648	0.07123	0.92877	0.06329	0.93665	0.05925	0.94042
0	9.4603	0.02793	0.97207	0.03098	0.96901	0.03106	0.96883	0	12.17246	0.07179	0.92821	0.06365	0.93629	0.05956	0.94011
0	9.518015	0.02818	0.97182	0.03131	0.96868	0.03137	0.96852	0	12.17669	0.07194	0.92806	0.06375	0.9362	0.05965	0.94002
0	9.521689	0.0282	0.9718	0.03133	0.96866	0.03139	0.9685	0	12.21318	0.07325	0.92675	0.06459	0.93536	0.06037	0.93929
0	9.523441	0.02821	0.97179	0.03134	0.96865	0.0314	0.96849	0	12.24029	0.07424	0.92576	0.06522	0.93472	0.06092	0.93874
0	9.627265	0.0287	0.9713	0.03197	0.96802	0.03196	0.96792	0	12.30899	0.0768	0.9232	0.06687	0.93307	0.06235	0.9373
0	9.634263	0.02874	0.97126	0.03202	0.96797	0.032	0.96788	0	12.31491	0.07702	0.92298	0.06702	0.93292	0.06248	0.93717
0	9.659343	0.02886	0.97114	0.03217	0.96781	0.03214	0.96774	0	12.3221	0.0773	0.9227	0.0672	0.93274	0.06263	0.93702
0	9.661699	0.02888	0.97112	0.03219	0.9678	0.03216	0.96773	0	12.3231	0.07733	0.92267	0.06722	0.93272	0.06265	0.937
0	9.668066	0.02891	0.97109	0.03223	0.96776	0.03219	0.96769	0	12.39222	0.08	0.92	0.06897	0.93097	0.06416	0.93548
0	9.703249	0.0291	0.9709	0.03245	0.96753	0.03239	0.96749	0	12.42939	0.08147	0.91853	0.06993	0.93	0.065	0.93463
0	9.706591	0.02911	0.97089	0.03248	0.96751	0.03241	0.96747	0	12.45402	0.08245	0.91755	0.07058	0.92935	0.06556	0.93406
0	9.741885	0.02931	0.97069	0.03271	0.96728	0.03262	0.96726	0	12.5098	0.08472	0.91528	0.07209	0.92784	0.06687	0.93275
0	9.7428	0.02931	0.97069	0.03271	0.96728	0.03262	0.96726	0	12.55014	0.08638	0.91362	0.07322	0.92672	0.06784	0.93177
0	9.776665	0.02951	0.97049	0.03294	0.96705	0.03282	0.96705	0	12.57123	0.08726	0.91274	0.07381	0.92612	0.06836	0.93124
0	9.779263	0.02952	0.97048	0.03295	0.96703	0.03284	0.96704	0	12.59401	0.08822	0.91178	0.07447	0.92546	0.06893	0.93067
0	9.815075	0.02973	0.97027	0.03319	0.96679	0.03306	0.96682	0	12.65097	0.09065	0.90935	0.07614	0.92379	0.07038	0.92921
0	9.820517	0.02976	0.97024	0.03323	0.96676	0.03309	0.96679	0	12.70497	0.09298	0.90702	0.07777	0.92215	0.0718	0.92778
0	9.843455	0.0299	0.9701	0.03339	0.9666	0.03323	0.96665	0	12.73969	0.0945	0.9055	0.07885	0.92107	0.07273	0.92684
0	9.854903	0.02997	0.97003	0.03347	0.96652	0.0333	0.96657	0	12.75345	0.09511	0.90489	0.07929	0.92064	0.07311	0.92646
0	9.876384	0.03011	0.96989	0.03362	0.96637	0.03343	0.96644	0	12.77124	0.0959	0.9041	0.07985	0.92007	0.0736	0.92596
0	9.880382	0.03013	0.96987	0.03364	0.96634	0.03346	0.96642	0	12.78645	0.09657	0.90343	0.08034	0.91958	0.07403	0.92554
0	9.916197	0.03036	0.96964	0.0339	0.96609	0.03368	0.96619	0	12.83125	0.09858	0.90142	0.0818	0.91812	0.0753	0.92426
0	9.940377	0.03052	0.96948	0.03407	0.96591	0.03384	0.96603	0	12.88883	0.10119	0.89881	0.08374	0.91618	0.07698	0.92256
0	9.960549	0.03066	0.96934	0.03422	0.96577	0.03397	0.9659	0	12.90275	0.10182	0.89818				

Queue Pos	Av. Serve Time	2'4"2		2'6"2		2'8"2		Queue Pos	Av. Serve Time	2'4"2		2'6"2		2'8"2	
		Reneg	Stay	Reneg	Stay	Reneg	Stay			Reneg	Stay	Reneg	Stay	Reneg	Stay
0	13.90871	0.14856	0.85144	0.13062	0.86924	0.11839	0.88085	1	1.21916	0.06164	0.93836	0.06234	0.9377	0.06162	0.9383
0	13.91741	0.14894	0.85106	0.13114	0.86872	0.11886	0.88038	1	1.227772	0.06229	0.93771	0.06297	0.93706	0.06225	0.93767
0	13.92297	0.14919	0.85081	0.13147	0.86839	0.11916	0.88008	1	1.263536	0.06495	0.93505	0.06558	0.93446	0.06483	0.93509
0	13.9235	0.14921	0.85079	0.1315	0.86836	0.11919	0.88005	1	1.296849	0.06738	0.93262	0.06796	0.93208	0.06718	0.93273
0	13.93042	0.14951	0.85049	0.13192	0.86794	0.11956	0.87967	1	1.314351	0.06864	0.93136	0.06919	0.93085	0.06839	0.93152
0	13.93161	0.14956	0.85044	0.13199	0.86787	0.11963	0.87961	1	1.367358	0.07234	0.92766	0.07281	0.92723	0.07196	0.92795
0	13.93261	0.14961	0.85039	0.13205	0.86781	0.11968	0.87955	1	1.389785	0.07386	0.92614	0.07429	0.92575	0.07343	0.92648
0	13.94035	0.14994	0.85006	0.13252	0.86734	0.1201	0.87913	1	1.393641	0.07412	0.92588	0.07454	0.9255	0.07368	0.92623
0	13.99535	0.15232	0.84768	0.1359	0.86396	0.12314	0.87607	1	1.458073	0.07831	0.92169	0.07864	0.9214	0.07772	0.92219
0	14.02133	0.15343	0.84657	0.13752	0.86233	0.12461	0.87459	1	1.554167	0.08406	0.91594	0.08428	0.91576	0.08328	0.91663
0	14.06906	0.15545	0.84455	0.14056	0.85928	0.12736	0.87182	1	1.581669	0.0856	0.9144	0.0858	0.91424	0.08477	0.91514
0	14.07182	0.15557	0.84443	0.14074	0.85911	0.12752	0.87166	1	1.603736	0.08679	0.91321	0.08698	0.91306	0.08593	0.91397
0	14.11578	0.1574	0.8426	0.14361	0.85623	0.13012	0.86904	1	1.656304	0.08952	0.91048	0.08968	0.91036	0.08859	0.91131
0	14.12767	0.15789	0.84211	0.1444	0.85545	0.13083	0.86832	1	1.709784	0.09212	0.90788	0.09227	0.90777	0.09114	0.90877
0	14.12951	0.15797	0.84203	0.14452	0.85532	0.13095	0.86821	1	1.751102	0.09401	0.90599	0.09416	0.90588	0.093	0.90691
0	14.15095	0.15885	0.84115	0.14595	0.85389	0.13224	0.8669	1	1.774818	0.09505	0.90495	0.09521	0.90484	0.09403	0.90588
0	14.18165	0.1601	0.8399	0.14803	0.85181	0.13413	0.86501	1	1.778995	0.09523	0.90477	0.09539	0.90466	0.09421	0.9057
0	14.23521	0.16225	0.83775	0.15171	0.84812	0.13749	0.86162	1	1.79124	0.09576	0.90424	0.09591	0.90413	0.09472	0.90518
0	14.23985	0.16243	0.83757	0.15204	0.8478	0.13779	0.86132	1	1.79599	0.09596	0.90404	0.09611	0.90393	0.09492	0.90499
0	14.25166	0.1629	0.8371	0.15287	0.84697	0.13855	0.86056	1	1.837687	0.09767	0.90233	0.09784	0.90221	0.09662	0.90329
0	14.30534	0.165	0.835	0.15668	0.84314	0.14204	0.85704	1	1.875189	0.09913	0.90087	0.09932	0.90073	0.09807	0.90183
0	14.33038	0.16597	0.83403	0.1585	0.84133	0.1437	0.85537	1	1.895191	0.09988	0.90012	0.10008	0.99997	0.09882	0.90108
0	14.34735	0.16661	0.83339	0.15974	0.84009	0.14484	0.85422	1	1.916264	0.10065	0.89935	0.10086	0.89918	0.0996	0.90031
0	14.37902	0.16781	0.83219	0.16208	0.83775	0.14699	0.85206	1	1.920539	0.1008	0.8992	0.10102	0.89903	0.09975	0.90016
0	14.38295	0.16796	0.83204	0.16237	0.83745	0.14726	0.85179	1	1.920958	0.10082	0.89918	0.10104	0.89901	0.09977	0.90014
0	14.40585	0.16882	0.83118	0.16408	0.83574	0.14884	0.8502	1	1.948707	0.10179	0.89821	0.10203	0.89801	0.10075	0.89916
0	14.47272	0.17127	0.82873	0.16919	0.83063	0.15356	0.84545	1	2.019404	0.10414	0.89586	0.10444	0.89561	0.10312	0.89679
0	14.51446	0.17276	0.82724	0.17245	0.82736	0.15658	0.84241	1	2.041779	0.10484	0.89516	0.10516	0.89489	0.10383	0.89608
0	14.5621	0.17443	0.82557	0.17624	0.82357	0.1601	0.83886	1	2.08194	0.10606	0.89394	0.10641	0.89364	0.10506	0.89485
0	14.59188	0.17546	0.82454	0.17864	0.82116	0.16234	0.83661	1	2.106387	0.10677	0.89323	0.10715	0.8929	0.10579	0.89412
0	14.5931	0.1755	0.8245	0.17874	0.82106	0.16243	0.83651	1	2.176828	0.10872	0.89128	0.10917	0.89088	0.10778	0.89213
0	14.60191	0.1758	0.8242	0.17946	0.82035	0.1631	0.83584	1	2.240825	0.11037	0.88963	0.11089	0.88916	0.10948	0.89043
0	14.6023	0.17582	0.82418	0.17949	0.82031	0.16313	0.83581	1	2.242658	0.11041	0.88959	0.11093	0.88911	0.10953	0.89038
0	14.66652	0.17797	0.82203	0.1848	0.815	0.1681	0.83081	1	2.273371	0.11117	0.88883	0.11172	0.88833	0.1103	0.88961
0	14.69159	0.1788	0.8212	0.1869	0.81289	0.17007	0.82882	1	2.275164	0.11121	0.88879	0.11177	0.88828	0.11035	0.88956
0	14.70629	0.17927	0.82073	0.18815	0.81165	0.17124	0.82765	1	2.306799	0.11197	0.88803	0.11256	0.88749	0.11113	0.88878
0	14.7401	0.18036	0.81964	0.19104	0.80875	0.17396	0.82491	1	2.335238	0.11263	0.88737	0.11325	0.8868	0.11181	0.8881
0	14.74658	0.18057	0.81943	0.19159	0.8082	0.17449	0.82438	1	2.370476	0.11343	0.88657	0.11409	0.88596	0.11264	0.88727
0	14.78353	0.18173	0.81827	0.1948	0.80499	0.17751	0.82134	1	2.379081	0.11362	0.88638	0.11429	0.88576	0.11284	0.88707
0	14.81191	0.18261	0.81739	0.19729	0.80249	0.17987	0.81896	1	2.379967	0.11364	0.88636	0.11431	0.88574	0.11286	0.88705
0	14.81527	0.18271	0.81729	0.19759	0.8022	0.18015	0.81868	1	2.391959	0.11391	0.88609	0.11458	0.88546	0.11313	0.88678
0	14.84576	0.18364	0.81636	0.2003	0.79949	0.18272	0.8161	1	2.408318	0.11427	0.88573	0.11496	0.88509	0.1135	0.88641
0	14.86915	0.18434	0.81566	0.20239	0.79739	0.18471	0.81409	1	2.443252	0.11502	0.88498	0.11575	0.8843	0.11428	0.88563
0	14.88426	0.18479	0.81521	0.20376	0.79602	0.18601	0.81279	1	2.464808	0.11548	0.88452	0.11622	0.88383	0.11475	0.88516
0	14.99408	0.18794	0.81206	0.21387	0.7859	0.19568	0.80306	1	2.505408	0.11632	0.88368	0.11711	0.88294	0.11562	0.88429
1	0.030402	0.0044	0.9956	0.00233	0.99768	0.00191	0.99809	1	2.526874	0.11676	0.88324	0.11756	0.88249	0.11607	0.88384
1	0.075089	0.00486	0.99514	0.00277	0.99723	0.00232	0.99767	1	2.534741	0.11692	0.88308	0.11773	0.88232	0.11624	0.88367
1	0.108158	0.00525	0.99475	0.00316	0.99684	0.00268	0.99731	1	2.537883	0.11698	0.88302	0.1178	0.88225	0.1163	0.88361
1	0.118258	0.00538	0.99462	0.00329	0.99672	0.0028	0.99719	1	2.547546	0.11717	0.88283	0.118	0.88205	0.11651	0.88341
1	0.132046	0.00556	0.99444	0.00347	0.99653	0.00298	0.99701	1	2.601956	0.11825	0.88175	0.11913	0.88092	0.11762	0.88229
1	0.156683	0.00591	0.99409	0.00383	0.99618	0.00331	0.99668	1	2.634596	0.11889	0.88111	0.11979	0.88026	0.11828	0.88163
1	0.161301	0.00597	0.99403	0.0039	0.99611	0.00338	0.99661	1	2.655143	0.11929	0.88071	0.12021	0.87984	0.11869	0.88123
1	0.215364	0.00684	0.99316	0.00481	0.99552	0.00425	0.99573	1	2.687157	0.1199	0.8801	0.12085	0.8792	0.11932	0.88059
1	0.281147	0.0081	0.9919	0.00618	0.99383	0.00558	0.9944	1	2.747963	0.12105	0.87895	0.12205	0.87801	0.1205	0.87941
1	0.301886	0.00855	0.99145	0.00667	0.99334	0.00607	0.99391	1	2.749009	0.12107	0.87893	0.12207	0.87799	0.12052	0.87939
1	0.312307	0.00879	0.99121	0.00694	0.99307	0.00632	0.99366	1	2.76642	0.1214	0.8786	0.12241	0.87765	0.12086	0.87905
1	0.328435	0.00918	0.99082	0.00736	0.99265	0.00674	0.99324	1	2.825936	0.12252	0.87748	0.12356	0.87649	0.122	0.87791
1	0.443163	0.01246	0.98754	0.01105	0.98897	0.01039	0.98958	1	2.884445	0.12361	0.87639	0.12469	0.87536	0.12311	0.87768
1	0.506047	0.01473	0.98527	0.0136	0.98642	0.01294	0.98703	1	2.923919	0.12434	0.87566	0.12545	0.8746	0.12386	0.87605
1	0.528754	0.01564	0.98436	0.01462	0.9854	0.01396	0.98601	1	2.958745	0.12499	0.87501	0.12612	0.87393	0.12452	0.87539
1	0.53205	0.01577	0.98423	0.01477	0.98525	0.01411	0.98585	1	2.980592	0.1254	0.8746	0.12654	0.87351	0.12494	0.87498
1	0.54442	0.01629	0.98371	0.01535	0.98466	0.01469	0.98527	1	2.989258	0.12556	0.87444	0.12671	0.87335	0.1251	0.87481
1	0.555892	0.01679	0.98321	0.01591	0.98411	0.01524	0.98472	1	3.136601	0.12835	0.87165	0.12957	0.87048	0.12792	0.87199
1	0.578765	0.01781	0.98219	0.01705	0.98297	0.01639	0.98357	1	3.149421	0.1286	0.8714	0.12982	0.87023	0.12817	0.87174
1	0.57927	0.01783	0.98217	0.01707	0.98295	0.01641	0.98354	1	3.168689	0.12897	0.87103				

Queue Pos	Av. Serve Time	2*4*2		2*6*2		2*8*2		Queue Pos	Av. Serve Time	2*4*2		2*6*2		2*8*2	
		Reneg	Stay	Reneg	Stay	Reneg	Stay			Reneg	Stay	Reneg	Stay	Reneg	Stay
1	4.334149	0.15785	0.84215	0.15936	0.8407	0.15721	0.84269	1	7.946815	0.3335	0.6665	0.43936	0.56068	0.44311	0.55643
1	4.370389	0.15904	0.84096	0.16055	0.83951	0.15838	0.84151	1	7.994453	0.33304	0.66696	0.44496	0.55508	0.44899	0.55055
1	4.420663	0.16073	0.83927	0.16224	0.83782	0.16005	0.83984	1	8.001725	0.33295	0.66705	0.44582	0.55422	0.44989	0.54965
1	4.632696	0.16837	0.83163	0.16988	0.83019	0.16758	0.83231	1	8.061327	0.33209	0.66791	0.45283	0.54721	0.45725	0.54228
1	4.653374	0.16917	0.83083	0.17067	0.8294	0.16836	0.83153	1	8.088686	0.3316	0.6684	0.45605	0.54399	0.46064	0.53889
1	4.655519	0.16925	0.83075	0.17075	0.82932	0.16844	0.83144	1	8.119846	0.33098	0.66902	0.45971	0.54033	0.46449	0.53504
1	4.666652	0.16968	0.83032	0.17118	0.82889	0.16887	0.83102	1	8.126324	0.33084	0.66916	0.46047	0.53957	0.46529	0.53424
1	4.689075	0.17055	0.82945	0.17205	0.82801	0.16973	0.83016	1	8.133633	0.33068	0.66932	0.46133	0.53871	0.46619	0.53333
1	4.751232	0.17303	0.82697	0.17452	0.82555	0.17217	0.82771	1	8.140154	0.33053	0.66947	0.46209	0.53794	0.46699	0.53253
1	4.930594	0.18065	0.81935	0.1821	0.81796	0.17968	0.82019	1	8.191259	0.32927	0.67073	0.46809	0.53195	0.4733	0.52621
1	5.075838	0.18733	0.81267	0.18877	0.81129	0.1863	0.81357	1	8.202847	0.32895	0.67105	0.46945	0.53059	0.47473	0.52478
1	5.126111	0.18976	0.81024	0.19119	0.80887	0.18871	0.81116	1	8.213755	0.32865	0.67135	0.47072	0.52931	0.47608	0.52344
1	5.213156	0.1941	0.8059	0.19554	0.80453	0.19303	0.80683	1	8.227426	0.32825	0.67175	0.47232	0.52771	0.47776	0.52175
1	5.286021	0.19787	0.80213	0.19932	0.80075	0.1968	0.80306	1	8.237505	0.32796	0.67204	0.4735	0.52653	0.479	0.52051
1	5.356418	0.20164	0.79836	0.2031	0.79696	0.20057	0.79928	1	8.243875	0.32776	0.67224	0.47425	0.52579	0.47978	0.51973
1	5.364282	0.20206	0.79794	0.20353	0.79653	0.201	0.79885	1	8.295775	0.32609	0.67391	0.4803	0.51973	0.48616	0.51334
1	5.419588	0.20511	0.79489	0.20661	0.79346	0.20407	0.79577	1	8.322647	0.32515	0.67485	0.48343	0.51661	0.48946	0.51004
1	5.458927	0.20732	0.79268	0.20885	0.79122	0.20631	0.79353	1	8.323266	0.32513	0.67487	0.4835	0.51653	0.48953	0.50997
1	5.501942	0.20978	0.79022	0.21135	0.78872	0.20881	0.79104	1	8.353008	0.32403	0.67597	0.48695	0.51308	0.49317	0.50632
1	5.527228	0.21124	0.78876	0.21284	0.78723	0.2103	0.78954	1	8.369334	0.3234	0.6766	0.48884	0.51119	0.49517	0.50433
1	5.527648	0.21127	0.78873	0.21286	0.7872	0.21032	0.78952	1	8.375272	0.32317	0.67683	0.48953	0.5105	0.49589	0.5036
1	5.551527	0.21267	0.78733	0.21429	0.78578	0.21175	0.78809	1	8.428352	0.32099	0.67901	0.49566	0.50437	0.50236	0.49713
1	5.578493	0.21426	0.78574	0.21592	0.78415	0.21338	0.78646	1	8.451377	0.31999	0.68001	0.4983	0.50172	0.50515	0.49434
1	5.622427	0.21689	0.78311	0.21861	0.78145	0.21608	0.78375	1	8.517344	0.31695	0.68305	0.50585	0.49418	0.51312	0.48636
1	5.671715	0.21989	0.78011	0.22171	0.77836	0.21918	0.78064	1	8.518565	0.31689	0.68311	0.50599	0.49404	0.51327	0.48622
1	5.676365	0.22017	0.77983	0.222	0.77806	0.21948	0.78035	1	8.520406	0.3168	0.6832	0.5062	0.49383	0.51349	0.486
1	5.676403	0.22018	0.77982	0.22201	0.77806	0.21948	0.78034	1	8.568576	0.31443	0.68557	0.51167	0.48836	0.51926	0.48021
1	5.742754	0.2243	0.7757	0.22629	0.77378	0.22378	0.77604	1	8.576286	0.31403	0.68597	0.51254	0.48748	0.52019	0.47929
1	5.771189	0.2261	0.7739	0.22817	0.7719	0.22567	0.77415	1	8.611491	0.31221	0.68779	0.51651	0.48351	0.52438	0.47509
1	5.79817	0.22781	0.77219	0.22997	0.7701	0.22748	0.77234	1	8.679242	0.30852	0.69148	0.52409	0.47593	0.53239	0.46707
1	5.807256	0.2284	0.7716	0.23058	0.76949	0.2281	0.77172	1	8.701789	0.30725	0.69275	0.5266	0.47342	0.53504	0.46442
1	5.84812	0.23103	0.76897	0.23336	0.7667	0.2309	0.76892	1	8.75618	0.3041	0.6959	0.5326	0.46742	0.54139	0.45807
1	5.918575	0.23564	0.76436	0.23828	0.76179	0.23585	0.76396	1	8.813918	0.30064	0.69936	0.53891	0.46111	0.54806	0.45139
1	5.973965	0.23932	0.76068	0.24225	0.75782	0.23985	0.75995	1	8.862072	0.29766	0.70234	0.54412	0.4559	0.55357	0.44588
1	5.982602	0.2399	0.7601	0.24288	0.75719	0.24049	0.75931	1	8.871178	0.29709	0.70291	0.5451	0.45492	0.55461	0.44484
1	5.997167	0.24088	0.75912	0.24394	0.75612	0.24156	0.75824	1	8.891166	0.29584	0.70416	0.54724	0.45278	0.55688	0.44258
1	6.018249	0.2423	0.7577	0.24549	0.75457	0.24313	0.75667	1	8.916543	0.29422	0.70578	0.54995	0.45007	0.55974	0.43971
1	6.044821	0.2441	0.7559	0.24747	0.7526	0.24512	0.75467	1	8.923645	0.29377	0.70623	0.5507	0.44932	0.56054	0.43891
1	6.114373	0.24884	0.75116	0.25274	0.74732	0.25045	0.74933	1	8.942144	0.29258	0.70742	0.55266	0.44735	0.56262	0.43683
1	6.117285	0.24904	0.75096	0.25297	0.7471	0.25068	0.74911	1	9.069342	0.28423	0.71577	0.56593	0.43409	0.57666	0.42278
1	6.147474	0.25112	0.74888	0.25531	0.74476	0.25305	0.74674	1	9.082992	0.28332	0.71668	0.56733	0.43269	0.57814	0.4213
1	6.16721	0.25248	0.74752	0.25685	0.74321	0.25461	0.74517	1	9.126027	0.28044	0.71956	0.57171	0.42831	0.58278	0.41666
1	6.245404	0.25791	0.74209	0.26309	0.73697	0.26094	0.73883	1	9.137115	0.27969	0.72031	0.57283	0.42718	0.58397	0.41548
1	6.25916	0.25886	0.74114	0.26421	0.73585	0.26207	0.7377	1	9.147199	0.27901	0.72099	0.57384	0.42617	0.58504	0.4144
1	6.277992	0.26017	0.73983	0.26575	0.73431	0.26364	0.73613	1	9.157576	0.27832	0.72168	0.57489	0.42512	0.58615	0.41329
1	6.307706	0.26225	0.73775	0.2682	0.73186	0.26613	0.73364	1	9.172844	0.27729	0.72271	0.57642	0.42359	0.58777	0.41167
1	6.326387	0.26355	0.73645	0.26976	0.73031	0.26771	0.73206	1	9.181466	0.27671	0.72329	0.57728	0.42273	0.58868	0.41076
1	6.338792	0.26441	0.73559	0.2708	0.72927	0.26876	0.731	1	9.224138	0.27383	0.72617	0.58151	0.4185	0.59316	0.40628
1	6.353312	0.26543	0.73457	0.27202	0.72804	0.27001	0.72976	1	9.26343	0.27117	0.72883	0.58536	0.41465	0.59724	0.4022
1	6.371332	0.26668	0.73332	0.27355	0.72652	0.27156	0.7282	1	9.270347	0.27071	0.72929	0.58604	0.41397	0.59795	0.40148
1	6.382639	0.26747	0.73253	0.27451	0.72555	0.27254	0.72722	1	9.311446	0.26794	0.73206	0.59001	0.40999	0.60216	0.39727
1	6.385843	0.2677	0.7323	0.27479	0.72528	0.27282	0.72694	1	9.341261	0.26593	0.73407	0.59287	0.40714	0.60519	0.39425
1	6.469237	0.27349	0.72651	0.28202	0.71804	0.28018	0.71956	1	9.411766	0.26122	0.73878	0.59952	0.40049	0.61223	0.38721
1	6.478463	0.27413	0.72587	0.28284	0.71723	0.28101	0.71873	1	9.435146	0.25967	0.74033	0.60169	0.39831	0.61453	0.3849
1	6.563537	0.27999	0.72001	0.29046	0.7096	0.2888	0.71094	1	9.440479	0.25932	0.74068	0.60219	0.39782	0.61505	0.38438
1	6.567712	0.28027	0.71973	0.29084	0.70922	0.28918	0.71055	1	9.448415	0.25879	0.74121	0.60292	0.39708	0.61583	0.38361
1	6.570481	0.28046	0.71954	0.2911	0.70897	0.28944	0.7103	1	9.449361	0.25873	0.74127	0.60301	0.397	0.61592	0.38351
1	6.72425	0.29076	0.70924	0.30547	0.69459	0.30415	0.69557	1	9.481267	0.25663	0.74337	0.60594	0.39407	0.61902	0.38041
1	6.744322	0.29206	0.70794	0.3074	0.69266	0.30612	0.69359	1	9.506502	0.25499	0.74501	0.60823	0.39177	0.62144	0.37799
1	6.747437	0.29227	0.70773	0.3077	0.69236	0.30643	0.69328	1	9.507776	0.2549	0.7451	0.60835	0.39166	0.62157	0.37786
1	6.852016	0.2989	0.7011	0.31794	0.68212	0.31695	0.68275	1	9.517934	0.25424	0.74576	0.60926	0.39074	0.62254	0.37689
1	6.863415	0.2996	0.7004	0.31908	0.68098	0.31811	0.68159	1	9.574777	0.25058	0.74942	0.61434	0.38566	0.62791	0.37152
1	6.874237	0.30026	0.69974	0.32016	0.6799	0.31922	0.68047	1	9.59206	0.24948	0.75052	0.61587	0.38413	0.62953	0.3699
1	6.891259	0.3013	0.6987	0.32187	0.67819	0.32098	0.67872	1	9.626396	0.24731	0.75269	0.61887	0.38113	0.6327	0.36673
1	6.895357	0.30154	0.69846	0.32228	0.67778	0.3214	0.								

Queue	Av. Serve	2*4*2		2*6*2		2*8*2		Queue	Av. Serve	2*4*2		2*6*2		2*8*2	
Pos	Time	Renege	Stay	Renege	Stay	Renege	Stay	Pos	Time	Renege	Stay	Renege	Stay	Renege	Stay
1	10.62301	0.19932	0.80068	0.69081	0.30918	0.70858	0.29087	1	13.07622	0.18166	0.81834	0.77392	0.22606	0.7952	0.20435
1	10.70412	0.19681	0.80319	0.69543	0.30455	0.71343	0.28602	1	13.08352	0.1817	0.8183	0.77404	0.22593	0.79533	0.20423
1	10.74147	0.19572	0.80428	0.6975	0.30248	0.71561	0.28384	1	13.19096	0.18241	0.81759	0.77582	0.22416	0.79717	0.20239
1	10.74327	0.19567	0.80433	0.6976	0.30238	0.71571	0.28374	1	13.19434	0.18244	0.81756	0.77588	0.2241	0.79723	0.20234
1	10.76132	0.19516	0.80484	0.69859	0.30139	0.71675	0.28271	1	13.21169	0.18256	0.81744	0.77615	0.22382	0.79751	0.20205
1	10.83024	0.19329	0.80671	0.70229	0.2977	0.72062	0.27883	1	13.26968	0.18296	0.81704	0.77706	0.22291	0.79845	0.20111
1	10.92145	0.19104	0.80896	0.707	0.29299	0.72556	0.2739	1	13.27165	0.18297	0.81703	0.77709	0.22288	0.79849	0.20108
1	10.99288	0.18943	0.81057	0.71055	0.28944	0.72928	0.27019	1	13.29804	0.18316	0.81684	0.7775	0.22248	0.7989	0.20066
1	11.0021	0.18923	0.81077	0.71099	0.28899	0.72975	0.26972	1	13.344	0.1835	0.8165	0.77819	0.22179	0.79962	0.19995
1	11.0546	0.18815	0.81185	0.71352	0.28647	0.73239	0.26708	1	13.34477	0.1835	0.8165	0.7782	0.22177	0.79963	0.19994
1	11.11652	0.18696	0.81304	0.71641	0.28357	0.73542	0.26405	1	13.34939	0.18353	0.81647	0.77827	0.22171	0.7997	0.19987
1	11.1205	0.18689	0.81311	0.71659	0.28339	0.73561	0.26386	1	13.42357	0.18408	0.81592	0.77935	0.22062	0.80082	0.19875
1	11.19833	0.18554	0.81446	0.7201	0.27988	0.73928	0.2602	1	13.51147	0.18475	0.81525	0.78058	0.2194	0.80209	0.19749
1	11.22504	0.18511	0.81489	0.72128	0.2787	0.7405	0.25897	1	13.52467	0.18486	0.81514	0.78076	0.21922	0.80227	0.1973
1	11.22956	0.18503	0.81497	0.72147	0.27851	0.74071	0.25876	1	13.59513	0.18541	0.81459	0.7817	0.21828	0.80324	0.19633
1	11.26243	0.18453	0.81547	0.7229	0.27709	0.74219	0.25728	1	13.5984	0.18543	0.81457	0.78174	0.21823	0.80329	0.19629
1	11.33488	0.1835	0.8165	0.72595	0.27403	0.74538	0.2541	1	13.6131	0.18555	0.81445	0.78193	0.21804	0.80349	0.19609
1	11.35226	0.18327	0.81673	0.72667	0.27332	0.74613	0.25335	1	13.65839	0.18591	0.81409	0.78252	0.21746	0.80409	0.1955
1	11.3576	0.1832	0.8168	0.72688	0.2731	0.74636	0.25312	1	13.67295	0.18603	0.81397	0.7827	0.21728	0.80428	0.19531
1	11.36312	0.18313	0.81687	0.72711	0.27287	0.74659	0.25289	1	13.67772	0.18607	0.81393	0.78276	0.21722	0.80434	0.19524
1	11.36325	0.18313	0.81687	0.72711	0.27287	0.7466	0.25288	1	13.70101	0.18625	0.81375	0.78305	0.21693	0.80464	0.19494
1	11.37091	0.18303	0.81697	0.72743	0.27255	0.74692	0.25256	1	13.71197	0.18634	0.81366	0.78319	0.21679	0.80478	0.1948
1	11.39889	0.18268	0.81732	0.72856	0.27142	0.7481	0.25138	1	13.72023	0.18641	0.81359	0.78329	0.21669	0.80488	0.1947
1	11.4278	0.18234	0.81766	0.72971	0.27027	0.7493	0.25018	1	13.78883	0.18697	0.81303	0.78411	0.21586	0.80574	0.19385
1	11.43507	0.18226	0.81774	0.72999	0.26999	0.7496	0.24988	1	13.85993	0.18755	0.81245	0.78494	0.21503	0.8066	0.19299
1	11.45437	0.18204	0.81796	0.73075	0.26923	0.75039	0.24909	1	13.87254	0.18766	0.81234	0.78509	0.21489	0.80674	0.19285
1	11.50106	0.18155	0.81845	0.73255	0.26743	0.75227	0.24722	1	13.87667	0.18769	0.81231	0.78513	0.21484	0.80679	0.1928
1	11.5243	0.18132	0.81868	0.73343	0.26655	0.75319	0.2463	1	13.92631	0.18811	0.81189	0.78569	0.21429	0.80737	0.19222
1	11.58346	0.18078	0.81922	0.73563	0.26435	0.75548	0.24401	1	13.9312	0.18815	0.81185	0.78574	0.21423	0.80742	0.19217
1	11.62494	0.18044	0.81956	0.73713	0.26285	0.75704	0.24245	1	13.94079	0.18823	0.81177	0.78585	0.21413	0.80753	0.19206
1	11.67306	0.18009	0.81991	0.73884	0.26114	0.75882	0.24068	1	13.96461	0.18843	0.81157	0.78611	0.21387	0.8078	0.19179
1	11.72151	0.17977	0.82023	0.74051	0.25947	0.76056	0.23894	1	13.97284	0.18849	0.81151	0.7862	0.21378	0.80789	0.1917
1	11.815	0.17925	0.82075	0.74363	0.25635	0.7638	0.2357	1	14.0041	0.18876	0.81124	0.78653	0.21344	0.80824	0.19135
1	11.81553	0.17925	0.82075	0.74365	0.25633	0.76382	0.23568	1	14.01264	0.18883	0.81117	0.78662	0.21335	0.80833	0.19126
1	11.85068	0.17909	0.82091	0.74478	0.2552	0.765	0.2345	1	14.01779	0.18887	0.81113	0.78668	0.2133	0.80839	0.1912
1	11.85714	0.17906	0.82094	0.74499	0.25499	0.76522	0.23429	1	14.09567	0.18953	0.81047	0.78749	0.21249	0.80922	0.19037
1	11.88554	0.17895	0.82105	0.74589	0.25409	0.76615	0.23335	1	14.13123	0.18983	0.81017	0.78784	0.21213	0.80959	0.19
1	11.91261	0.17885	0.82115	0.74673	0.25325	0.76703	0.23248	1	14.14998	0.18999	0.81001	0.78803	0.21195	0.80979	0.18981
1	12.01412	0.17858	0.82142	0.7498	0.25018	0.77021	0.2293	1	14.17202	0.19017	0.80983	0.78825	0.21173	0.81001	0.18959
1	12.036	0.17854	0.82146	0.75044	0.24954	0.77088	0.22863	1	14.1788	0.19023	0.80977	0.78831	0.21166	0.81008	0.18952
1	12.09279	0.17846	0.82154	0.75207	0.24791	0.77257	0.22694	1	14.18114	0.19025	0.80975	0.78834	0.21164	0.8101	0.1895
1	12.11583	0.17843	0.82157	0.75272	0.24726	0.77324	0.22627	1	14.25121	0.19084	0.80916	0.78901	0.21097	0.8108	0.18881
1	12.22974	0.17841	0.82159	0.75581	0.24417	0.77645	0.22307	1	14.26134	0.19093	0.80907	0.7891	0.21088	0.81089	0.18871
1	12.27643	0.17844	0.82156	0.75703	0.24295	0.77772	0.22181	1	14.27872	0.19108	0.80892	0.78926	0.21072	0.81106	0.18854
1	12.30162	0.17847	0.82153	0.75767	0.24231	0.77838	0.22114	1	14.33601	0.19156	0.80844	0.78979	0.21019	0.8116	0.188
1	12.30434	0.17847	0.82153	0.75774	0.24224	0.77846	0.22107	1	14.33641	0.19157	0.80843	0.78979	0.21019	0.81161	0.188
1	12.31674	0.17849	0.82151	0.75805	0.24192	0.77878	0.22074	1	14.33875	0.19159	0.80841	0.78981	0.21017	0.81163	0.18798
1	12.33462	0.17851	0.82149	0.7585	0.24148	0.77925	0.22028	1	14.36702	0.19182	0.80818	0.79006	0.20992	0.81189	0.18772
1	12.35525	0.17855	0.82145	0.75902	0.24096	0.77978	0.21975	1	14.36882	0.19199	0.80801	0.79024	0.20974	0.81207	0.18754
1	12.44407	0.17873	0.82127	0.76116	0.23881	0.782	0.21753	1	14.4064	0.19216	0.80784	0.79041	0.20957	0.81225	0.18736
1	12.45687	0.17876	0.82124	0.76146	0.23851	0.78232	0.21721	1	14.47972	0.19278	0.80722	0.79103	0.20894	0.81289	0.18672
1	12.49653	0.17887	0.82113	0.76239	0.23759	0.78327	0.21626	1	14.49603	0.19292	0.80708	0.79117	0.20881	0.81304	0.18658
1	12.5222	0.17895	0.82105	0.76297	0.237	0.78388	0.21566	1	14.54138	0.1933	0.8067	0.79154	0.20844	0.81342	0.18619
1	12.58543	0.17916	0.82084	0.76439	0.23559	0.78534	0.2142	1	14.56285	0.19348	0.80652	0.79172	0.20826	0.8136	0.18601
1	12.6385	0.17936	0.82064	0.76554	0.23444	0.78653	0.21301	1	14.6559	0.19426	0.80574	0.79245	0.20753	0.81436	0.18526
1	12.66416	0.17946	0.82054	0.76608	0.2339	0.7871	0.21244	1	14.69974	0.19463	0.80537	0.79278	0.2072	0.8147	0.18491
1	12.73502	0.17977	0.82023	0.76755	0.23243	0.78862	0.21093	1	14.71718	0.19477	0.80523	0.79291	0.20707	0.81484	0.18478
1	12.79039	0.18003	0.81997	0.76866	0.23132	0.78977	0.20978	1	14.7635	0.19516	0.80484	0.79325	0.20673	0.81519	0.18443
1	12.7965	0.18006	0.81994	0.76878	0.2312	0.78989	0.20966	1	14.7694	0.1952	0.8048	0.79329	0.20669	0.81524	0.18438
1	12.87754	0.18048	0.81952	0.77035	0.22963	0.79151	0.20804	1	14.78081	0.1953	0.8047	0.79338	0.2066	0.81532	0.1843
1	12.88424	0.18052	0.81948	0.77047	0.2295	0.79164	0.20791	1	14.84585	0.19584	0.80416	0.79384	0.20614	0.8158	0.18382
1	12.89441	0.18058	0.81942	0.77066	0.22931	0.79184	0.20771	1	14.84714	0.19585	0.80415	0.79385	0.20613	0.81581	0.18381
1	12.91129	0.18067	0.81933	0.77098	0.229	0.79216	0.20739	1	14.86331	0.19598	0.80402	0.79396	0.20602	0.81593	0.1837
1	12.96666	0.18099	0.81901	0.77199	0.22798	0.79321	0.20634	1	14.93061	0.19653	0.80347	0.79442	0.20556	0.81641	0.18322
1	13.04373	0.18145	0.81855	0.77336	0.22662	0.79462	0.20493	1	14.95153	0.1967	0.8033	0.79456	0.20542	0	

Appendix J

Implementation of Intelligence Modules in the Bank Simulation

The implementation of the intelligence modules into the bank simulation involved two steps, firstly using the intelligence modules with just the special customers, who had previously been under the control of human decision makers, and secondly applying the modules to all of the customers. The second part required the removal of the data collection framework that had been used in the earlier part of the study to collect decision making data, but required virtually no changes to the decision making modules. Here, only the code for the decision making modules will be presented, along with the updated attribute storage for the customer entities.

J1 : Customer Attributes

The use of the decision making modules required an extension to the stored attributes for the customer entities (the previous attributes are shown in Appendix B3, pB-3). As well as further integer attributes, the entities required real valued attributes. Additional code was added to the simulation to allow for this.

Attribute	Representation	Values
1	Customer Type	1 = Information, 2 = Transactions, 3 = Business, 4 = Currency, 5 = Special
2	Service Required	1 = Information, 2 = Transactions, 3 = Business, 4 = Currency, 0 = leave
3	Queue Joined	1 - 5 - possible values depend on service
4	Position in Queue when first joined	Integer (positive)
5	Decision Group	0 = None assigned, 1 = Position in Queue ≤ 3 , Av. Service Time < 4 2 = Position in Queue ≤ 3 , Av. Service Time ≥ 4 3 = Position in Queue > 3
Real 1	Simulation Time when Queue joined	Real (positive)
Real 2	Highest Average Service Time	Real (positive)
Real 3	Random Number	Real - range 0-1

J2: Decision Module Code

Initialising the Neural Networks

The neural networks in the code make use of the routines in the library unit *simnet* (see Appendix A2, pA-32). Each neural network to be used needs to be declared as type *nnet*.

```
{* intelligence module variables *}
infbalk,transbalk,busbalk,curbalk,renege:nnet;  {* nnet structures *}
```

The neural network structures are initialised at the beginning of the run, loading in the node biases and weights from file.

```
{** CUSTOMER INTELLIGENCE MODULES **}

procedure setup_neural_nets;
  {* load neural networks structures into memory *}
var
  c:char;
begin

  {* load network structure from file *}
  {* setup creates structure using linked lists, and stores in memory *}
  {* returns true if successful, false if an error occurs *}

  if not set_network('inf_i2.nod','inf_i2.wgt',infbalk) then
    begin
      screen_on(1);
      open_own;
      text_form(1,20,50,'Info Knowledge load failure',0,1,red,yellow);
      text_form(1,20,120,'Press an key to continue',0,1,red,yellow);
      c:=readkey;
      close_own;
    end;

  if not set_network('tra_ib2.nod','tra_ib2.wgt',transbalk) then
    begin
      screen_on(1);
      open_own;
      text_form(1,20,50,'Transaction Knowledge load failure',0,1,red,yellow);
      text_form(1,20,120,'Press an key to continue',0,1,red,yellow);
      c:=readkey;
      close_own;
    end;

  if not set_network('bus_i3.nod','bus_i3.wgt',busbalk) then
    begin
      screen_on(1);
      open_own;
      text_form(1,20,50,'Business Knowledge load failure',0,1,red,yellow);
```

```

text_form(1,20,120,'Press an key to continue',0,1,red,yellow);
c:=readkey;
close_own;
end;

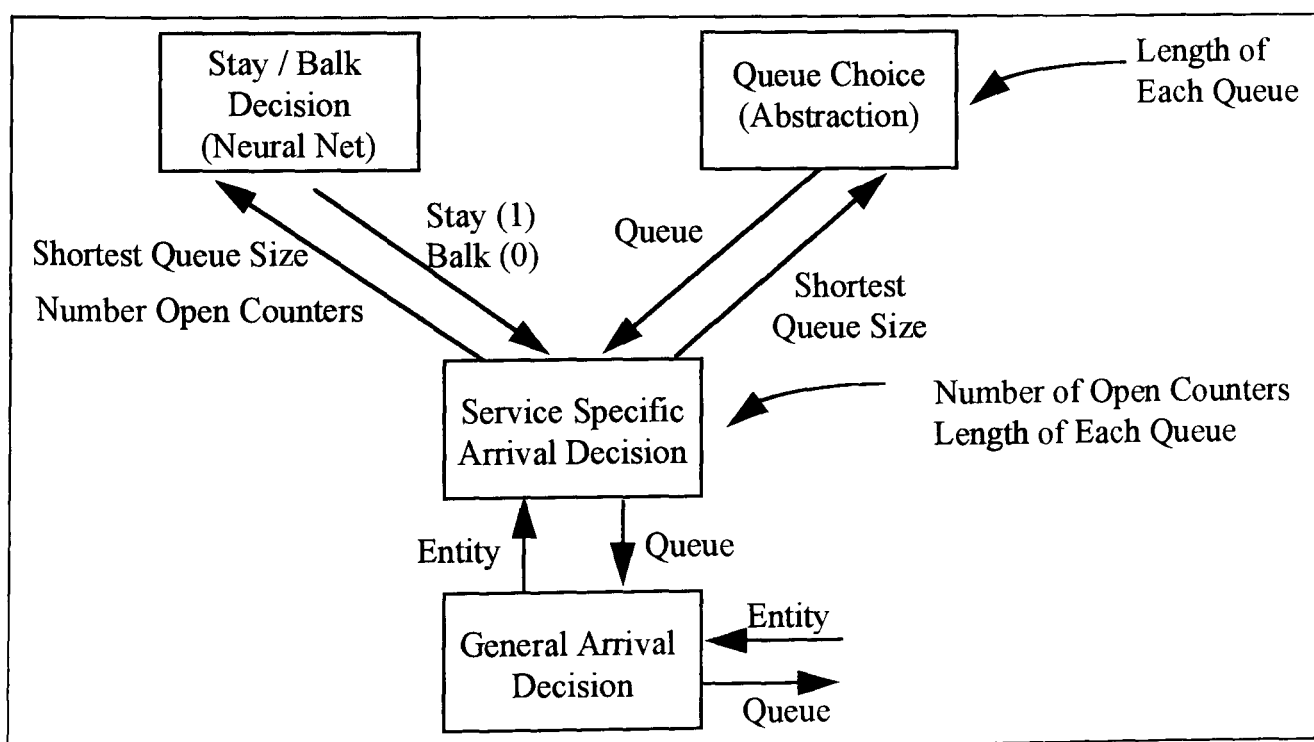
if not set_network('cur_ib2.nod','cur_ib2.wgt',curbalk) then
begin
screen_on(1);
open_own;
text_form(1,20,50,'Currency Knowledge load failure',0,1,red,yellow);
text_form(1,20,120,'Press an key to continue',0,1,red,yellow);
c:=readkey;
close_own;
end;

if not set_network('renege6.nod','renege6.wgt',renege) then
begin
screen_on(1);
open_own;
text_form(1,20,50,'Currency Knowledge load failure',0,1,red,yellow);
text_form(1,20,120,'Press an key to continue',0,1,red,yellow);
c:=readkey;
close_own;
end;
end;

```

Arrival Decisions - Stay in Bank or Balk

The diagram below shows the structure of the Stay / Balk decision modules. This is a generalised structure for all the service types (note that Information does not have a Queue Choice routine). Straight-lined arrows show exchange of information by parameter passing, while the curved arrows show direct access to data.



The code for the stay / balk decisions for all the service types is shown below. The basic structure of each one is as shown in the diagram. The each service accesses a different neural network model for the decision to stay or not. The choice of queue uses an abstraction approach in each case, but uses slightly different probability values for each service.

```

{***** INFORMATION ARRIVAL DECISION MODULE *****)}

function info_stay_or_balk(numcounters,shortest:integer):integer;
{ * decision module for information customers *}
{ * numcounters is number of counters open : 0-4 *}
{ * shortest is size of single queue + 1 person being served *}
{ * if shortest = 0 then there is at least one free counter *}
{ * applies rules and neural network model *}
{ * response is 1:stay, or 0:balk *}
var
  sc_numcounters:real;
  sc_shortest:real;
  rn:real;
  balk,stay:real;
  decision:integer;

begin
  if numcounters=0 then {** information closed **}
    decision:=0 {** leave **}
  else if shortest=0 then {** free counter, no queueing time **}
    decision:=1 {** stay **}
  else {** use neural network **}
    begin
      {** scale input data **}
      sc_numcounters:=scale(numcounters,1,4,0,1);
      sc_shortest:=scale(shortest-1,0,15,0,1);
      rn:=rnd(21); {** random number between 0 and 1 **}

      {** note shortest-1 since nn trained on length of single **}
      {** queue - the nn is only be used where no counters are free **}
      indata(infbalk,1,sc_numcounters); {** first input **}
      indata(infbalk,2,sc_shortest); {** second input **}
      indata(infbalk,3,rn); {** third input **}

      use_network(infbalk);
      stay:=outdata(infbalk,1); { * stay in bank *}
      balk:=outdata(infbalk,2); { * exit bank *}

      {** winner takes all, maximum output takes decision **}
      if stay >= balk then
        decision:=1
      else
        decision:=0;
      end;
      info_stay_or_balk:=decision;
    end;
  end;
end;

```



```

function information_arrival_decision:integer;
{** function returns 0 for balk, 1,2,3,4 for counter number and **}
{** 5 to join the single queue **}
{** counter joining rule : **}
{** if any counters are free - go to free counter nearest to door (left) **}
{** otherwise consult neural network for balk/stay **}
var
  counters_open,queue_size:integer;
  i:integer;
  decision:integer;
  counter_free:boolean;
begin
  {** find the number of information counters open **}
  counters_open:=0;
  for i:=1 to 4 do      {** four information desks **}
    if info[i]=open then
      inc(counters_open);

  {** find the length of the single queue **}
  queue_size:=size_of(info_q);
  {** if queue > 0, then all open counters must be serving **}

  counter_free:=false;
  if queue_size > 0 then
    queue_size:=queue_size+1 {** add one for counters occupied **}
  else {** no queue - are any of the counters free ? **}
    begin
      for i:=4 downto 1 do {** look at counters right to left **}
        if (info[i]=open) and (size_of(info_c[i])=0) then
          begin
            counter_free:=true; {** free counter found **}
            decision:=i; {** free counter furthest to left **}
          end;
        if not counter_free then
          queue_size:=queue_size+1; {** no free counter: add 1 to queue **}
        end;
      if not counter_free then {** no free counter - consult neural net **}
        begin
          decision:=info_stay_or_balk(counters_open,queue_size);
          if decision=1 then {** stay in bank **}
            decision:=5; {** indicate that customer joins single queue **}
          end;
        information_arrival_decision:=decision;
      end;

function trans_stay_or_balk(numcounters,shortest:integer):integer;
{** decision module for transactions customers **}
{** numcounters is number of counters open : 0-5 **}
{** shortest is size of single queue + 1 person being served **}
{** if shortest = 0 then there is at least one free counter **}
{** applies rules and neural network model **}
{** response is 1:stay, or 0:balk **}
var
  sc_numcounters:real;

```



```

sc_shortest:real;
rn:real;
balk,stay:real;
decision:integer;

begin
if numcounters=0 then  {** transactions closed **}
  decision:=0          {** leave **}
else if shortest=0 then {** free counter, no queueing time **}
  decision:=1          {** stay **}
else
  {** use neural network **}
  begin
  {** scale input data **}
  sc_numcounters:=scale(numcounters,1,5,0,1);
  sc_shortest:=scale(shortest,0,15,0,1);
  rn:=rnd(22); {** random number between 0 and 1 **}

  {** the nn is only be used where no counters are free **}
  indata(transbalk,1,sc_numcounters); {** first input **}
  indata(transbalk,2,sc_shortest);    {** second input **}
  indata(transbalk,3,rn);              {** third input **}

  use_network(transbalk);
  stay:=outdata(transbalk,1); {* stay in bank *}
  balk:=outdata(transbalk,2); {* exit bank *}

  {** winner takes all, maximum output takes decision **}
  if stay >= balk then
    decision:=1
  else
    decision:=0;
  end;
  trans_stay_or_balk:=decision;
end;

function trans_queue_select(shortest:integer):integer;
{** probability of selecting a particular shortest queue is based **}
{** on the queue preference weights **}
{** Prob(Queue i) = Weight(Queue i)/Sum of weights of shortest queues **}
{** where Queue i is one of the shortest queues **}
var
  qw:array[1..5] of real; {* queue preference weightings *}
  pq:array[0..5] of real; {* prob. of joining each queue *}
  sum_of_weights:real;
  i:integer;
  rn:real;
  cum_prob:real;
  decision:integer;
begin
  qw[1]:=1.667;  qw[2]:=1.022;  qw[3]:=1.282;
  qw[4]:=1.065;  qw[5]:=0.502;
  pq[0]:=0;

  {** find sum of weights of shortest queues **}
  sum_of_weights:=0;
  for i:=1 to 5 do
    if (transactions[i]=open) and
      (size_of(trans_c[i])+size_of(trans_q[i])=shortest) then

```

```

    sum_of_weights:=sum_of_weights+qw[i];

    ** find cumulative probabilities of selection **
    cum_prob:=0;
    for i:=1 to 5 do
        begin
            if (transactions[i]=open) and
                (size_of(trans_c[i])+size_of(trans_q[i])=shortest) then
                begin
                    cum_prob:=cum_prob+(qw[i]/sum_of_weights); ** cumulative probabilities **
                    pq[i]:=cum_prob;
                end
            else
                pq[i]:=-1; ** queue not eligible **
            end;
        end;

    ** find queue choice **
    rn:=rnd(25);
    decision:=0;
    repeat
        inc(decision)
    until rn<pq[decision];

    trans_queue_select:=decision;
end;

function transaction_arrival_decision:integer;
** function returns 0 for balk, 1,2,3,4,5 for queue number **
** counter joining rule : **
** queue selected on basis of shortest, with preference weightings **
** routines also copes with free counters **
var
    counters_open,queue_size:integer;
    i:integer;
    decision:integer;
    counter_free:boolean;
begin
    ** find the number of transaction counters open **
    counters_open:=0;
    for i:=1 to 5 do ** four information desks **
        if transactions[i]=open then
            inc(counters_open);

    ** find the length of the shortest queue **
    queue_size:=9999;
    for i:=1 to 5 do
        if transactions[i]=open then
            if size_of(trans_q[i])+size_of(trans_c[i]) < queue_size then
                queue_size:=size_of(trans_q[i])+size_of(trans_c[i]);

    ** if queue > 0, then all open counters must be serving **
    decision:=trans_stay_or_balk(counters_open,queue_size);
    if decision>0 then ** stay in bank - select queue**
        decision:=trans_queue_select(queue_size);

    transaction_arrival_decision:=decision;
end;

```

```

function bus_stay_or_balk(numcounters,shortest:integer):integer;
{ * decision module for business customers * }
{ * numcounters is number of counters open : 0-2 * }
{ * shortest is size of single queue + 1 person being served * }
{ * if shortest = 0 then there is at least one free counter * }
{ * applies rules and neural network model * }
{ * response is 1:stay, or 0:balk * }
var
  sc_numcounters:real;
  sc_shortest:real;
  rn:real;
  balk,stay:real;
  decision:integer;

begin
  if numcounters=0 then { ** business closed ** }
    decision:=0 { ** leave ** }
  else if shortest=0 then { ** free counter, no queuing time ** }
    decision:=1 { ** stay ** }
  else { ** use neural network ** }
    begin
      { ** scale input data ** }
      sc_numcounters:=scale(numcounters,1,2,0,1);
      sc_shortest:=scale(shortest,0,15,0,1);
      rn:=rnd(23); { ** random number between 0 and 1 ** }

      { ** the nn is only be used where no counters are free ** }
      indata(busbalk,1,sc_numcounters); { ** first input ** }
      indata(busbalk,2,sc_shortest); { ** second input ** }
      indata(busbalk,3,rn); { ** third input ** }

      use_network(busbalk);
      stay:=outdata(busbalk,1); { * stay in bank * }
      balk:=outdata(busbalk,2); { * exit bank * }

      { ** winner takes all, maximum output takes decision ** }
      if stay >= balk then
        decision:=1
      else
        decision:=0;
      end;
      bus_stay_or_balk:=decision;
    end;

function bus_queue_select(shortest:integer):integer;
{ ** select queue on basis of shortest ** }
{ ** if both counters open, and queues the same length, then ** }
{ ** use preference probabilities ** }
var
  decision:integer;
begin
  if ((business[1]=open) and (size_of(bus_q[1])+size_of(bus_c[1])=shortest))
  and ((business[2]<>open) or (size_of(bus_q[2])+size_of(bus_c[2])>shortest)) then
    decision:=1
  else if ((business[1]<>open) or (size_of(bus_q[1])+size_of(bus_c[1])>shortest))
  and ((business[2]=open) and (size_of(bus_q[2])+size_of(bus_c[2])=shortest)) then

```



```

    decision:=2
else if rnd(26)<=0.714 then
    decision:=1 {** both open & shortest, select queue 1 **}
else
    decision:=2; {** both open & shortest, select queue 2 **}

    bus_queue_select:=decision;
end;

function business_arrival_decision:integer;
{** function returns 0 for balk, 1,2,3,4,5 for queue number **}
{** counter joining rule : **}
{** queue selected on basis of shortest, with preference weightings **}
{** routines also copes with free counters **}
var
    counters_open,queue_size:integer;
    i:integer;
    decision:integer;
    counter_free:boolean;
begin
    {** find the number of transaction counters open **}
    counters_open:=0;
    for i:=1 to 2 do {** four information desks **}
        if business[i]=open then
            inc(counters_open);

    {** find the length of the shortest queue **}
    queue_size:=9999;
    for i:=1 to 2 do
        if business[i]=open then
            if size_of(bus_q[i])+size_of(bus_c[i]) < queue_size then
                queue_size:=size_of(bus_q[i])+size_of(bus_c[i]);

    {** if queue > 0, then all open counters must be serving **}
    decision:=bus_stay_or_balk(counters_open,queue_size);
    if decision>0 then {** stay in bank - select queue**}
        decision:=bus_queue_select(queue_size);

    business_arrival_decision:=decision;
end;

function cur_stay_or_balk(numcounters,shortest:integer):integer;
{** decision module for currency customers *}
{** numcounters is number of counters open : 0-2 *}
{** shortest is size of single queue + 1 person being served *}
{** if shortest = 0 then there is at least one free counter *}
{** applies rules and neural network model *}
{** response is 1:stay, or 0:balk *}
var
    sc_numcounters:real;
    sc_shortest:real;
    rn:real;
    balk,stay:real;
    decision:integer;
begin
    if numcounters=0 then {** currency closed **}

```

```

    decision:=0      {** leave **}
else if shortest=0 then {** free counter, no queueing time **}
    decision:=1      {** stay **}
else                {** use neural network **}
    begin
    {** scale input data **}
        sc_numcounters:=scale(numcounters,1,2,0,1);
        sc_shortest:=scale(shortest,0,15,0,1);
        rn:=rnd(24); {** random number between 0 and 1 **}

        {** the nn is only be used where no counters are free **}
        indata(curbalk,1,sc_numcounters); {** first input **}
        indata(curbalk,2,sc_shortest);    {** second input **}
        indata(curbalk,3,rn);             {** third input **}

        use_network(curbalk);
        stay:=outdata(curbalk,1); {* stay in bank *}
        balk:=outdata(curbalk,2); {* exit bank *}

    {** winner takes all, maximum output takes decision **}
        if stay >= balk then
            decision:=1
        else
            decision:=0;
        end;
        cur_stay_or_balk:=decision;
    end;

function cur_queue_select(shortest:integer):integer;
{** select queue on basis of shortest **}
{** if both counters open, and queues the same length, then **}
{** use preference probabilities **}
var
    decision:integer;
begin
    if ((currency[1]=open) and (size_of(curr_q[1])+size_of(curr_c[1])=shortest))
    and ((currency[2]<>open) or (size_of(curr_q[2])+size_of(curr_c[2])>shortest)) then
        decision:=1
    else if ((currency[1]<>open) or (size_of(curr_q[1])+size_of(curr_c[1])>shortest))
    and ((currency[2]=open) and (size_of(curr_q[2])+size_of(curr_c[2])=shortest)) then
        decision:=2
    else if rnd(27)<=0.714 then
        decision:=1 {** both open & shortest, select queue 1 **}
    else
        decision:=2; {** both open & shortest, select queue 2 **}

    cur_queue_select:=decision;
end;

function currency_arrival_decision:integer;
{** function returns 0 for balk, 1,2,3,4,5 for queue number **}
{** counter joining rule : **}
{** queue selected on basis of shortest, with preference weightings **}
{** routines also copes with free counters **}
var
    counters_open.queue_size:integer;
    i:integer;
    decision:integer;

```



```

counter_free:boolean;
begin
  /** find the number of transaction counters open **}
  counters_open:=0;
  for i:=1 to 2 do      /** four information desks **}
    if currency[i]=open then
      inc(counters_open);

  /** find the length of the shortest queue **}
  queue_size:=9999;
  for i:=1 to 2 do
    if currency[i]=open then
      if size_of(curr_q[i])+size_of(curr_c[i]) < queue_size then
        queue_size:=size_of(curr_q[i])+size_of(curr_c[i]);

  /** if queue > 0, then all open counters must be serving **}
  decision:=cur_stay_or_balk(counters_open,queue_size);
  if decision>0 then /** stay in bank - select queue**}
    decision:=cur_queue_select(queue_size);

  currency_arrival_decision:=decision;
end;

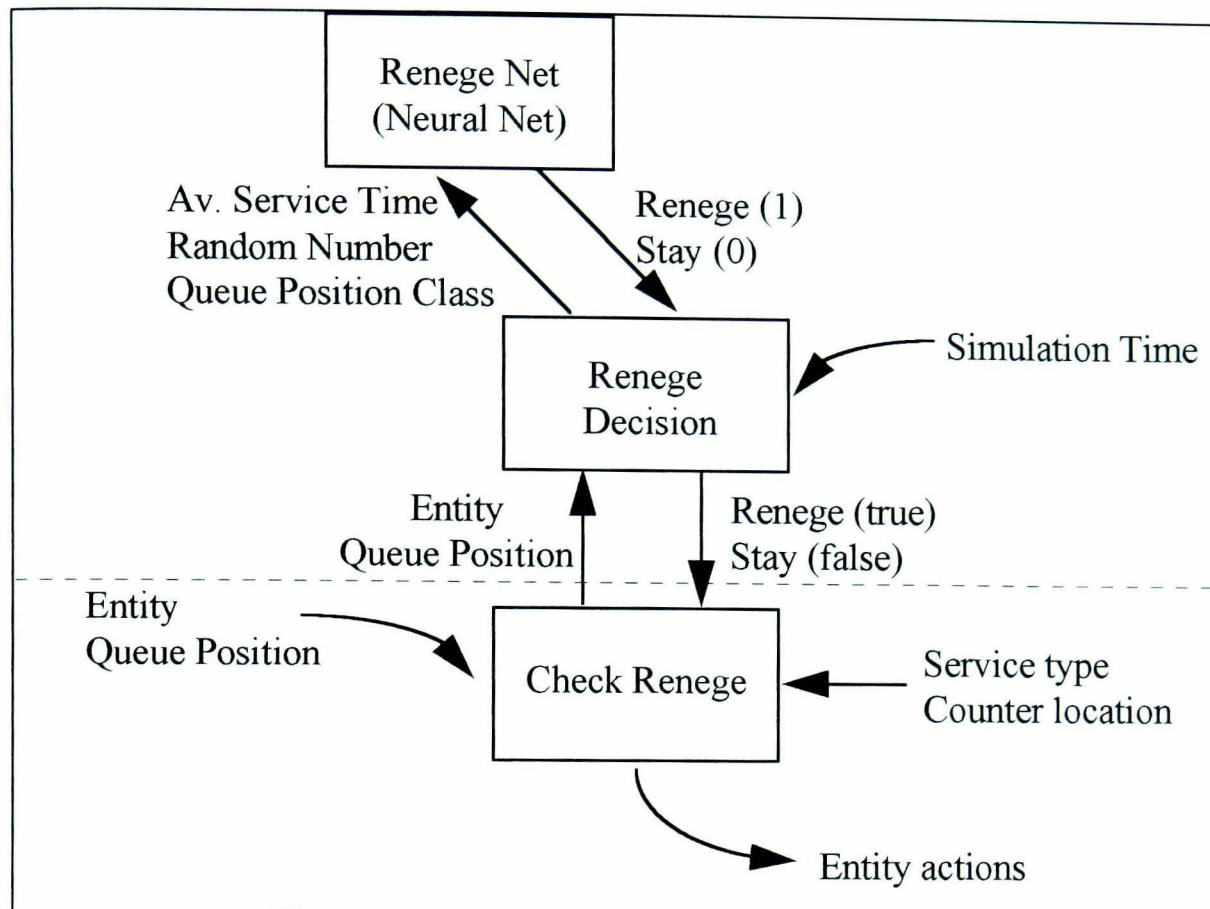
function customer_arrival_decision(customer:entity):integer;
/** identify customer service requirement, and call appropriate **}
/** decision making sub-module **}
begin
  if attribute(customer,2)=1 then /** information **}
    customer_arrival_decision:=information_arrival_decision
  else if attribute(customer,2)=2 then /** transactions **}
    customer_arrival_decision:=transaction_arrival_decision
  else if attribute(customer,2)=3 then /** business **}
    customer_arrival_decision:=business_arrival_decision
  else if attribute(customer,2)=4 then /** currency **}
    customer_arrival_decision:=currency_arrival_decision
end;

```

Renege Decision

The Renege decision module is made up two main parts. The *renege_decision* function calculates the decision parameters from the attributes of the entity, determines the decision group, and determines whether there is a possibility of renegeing. If renegeing is a possibility, then the *renege_net* function is called to resolve the decision. A third procedure is *check_renege* which is an interface between the simulation program and the decision module, and so it does not truly count as a part of the decision module. It is called when the state a particular queue changes, and then goes through each customer in the queue to check their renege decision. When the decision is returned, it controls the actions of the customer.

The structure of the decision module, with the interface, is shown below.



The program code for the routines is below. Note that all of the extra entity attributes shown in J1 are used in the *renege_decision* function.

```

{** RENEGE DECISION MODULE **}

function renege_net(q_pos_class:integer;av_serve,m:real):integer;
var
  sc_av_serve:real;
  renege_prob,stay:real;

begin
  sc_av_serve:=scale(av_serve,0,15,0,1);
  {** the nn is only be used where no counters are free **}
  indata(renege,1,q_pos_class); {** first input **}
  indata(renege,2,sc_av_serve); {** second input **}
  indata(renege,3,m); {** third input **}

  use_network(renege);
  renege_prob:=outdata(renege,1); {* exit bank *}
  stay:=outdata(renege,2); {* stay in bank *}

  {** rescale renege_prob. so that sum of probabilities adds to 1 **}
  {** stay prob is implicitly re-scaled too. but is not used in any **}
  {** further calculations **}
  
```

```

renege_prob:=renege_prob/(renege_prob+stay);

{** winner takes all, maximum output takes decision **}
if rn <= renege_prob then
  renege_net:=1
else
  renege_net:=0;

end;

function renege_decision(element:entity;q_pos:integer):boolean;
var
  av_serve_time:real;
  decision:integer;
begin
  {** average service time while in queue **}
  {** this is current simulation time - time when queue joined **}
  {** divided by **}
  {** position in queue when joined - current position (minimum 1) **}
  av_serve_time:=(time-real_attribute(element,1))/
    max((attribute(element,4))-q_pos,1);

  if av_serve_time <= real_attribute(element,2) then
    decision:=0 {** only renege if av. service time has increased **}
  else if q_pos = 1 then
    decision:=0 {** do not renege when next to be served **}
  else if (q_pos<=3) and (av_serve_time<4.0) then
    decision:=0 {** do not renege when close to front and low av. serve **}
  else if (q_pos<=3) then {** close to front, but longer av. serve time **}
    begin
      if attribute(element,5)<>2 then {** generate new rn **}
        begin
          set_real_attribute(element,3,rnd(27));
          set_attribute(element,5,2)
        end;
      decision:=renege_net(0,av_serve_time,real_attribute(element,3))
    end
  else {** note close to front, and av_serve_time increased **}
    begin
      if attribute(element,5)<>3 then {** generate new rn **}
        begin
          set_real_attribute(element,3,rnd(27));
          set_attribute(element,5,3)
        end;
      decision:=renege_net(1,av_serve_time,real_attribute(element,3))
    end;

  {** update maximum average serve time **}
  if av_serve_time>=real_attribute(element,2) then
    set_real_attribute(element,2,av_serve_time);

  if decision=0 then
    renege_decision:=false {** do not renege **}
  else
    renege_decision:=true {** renege **}
end;

```

```

procedure check_renege(c,q:integer);
{** called when a customer service is about to finish **}
{** check the queue to see if anyone reneges before this **}
{** happens. c is counter type, q is queue number**}
var
  element:entity;
  queue:sets;
  i:integer;
  num_in_q:integer;
  q_pos:integer;
begin
  queue:=queue_set(c,q);
  num_in_q:=size_of(queue);

  {** check each person in queue from last to first **}
  {** order prevents side effects from other customers **}
  {** reneging **}
  for i:= num_in_q downto 1 do
    begin
      element:=identity(i,queue);
      q_pos:=position(element,queue);
      if renege_decision(element,q_pos) then
        begin
          case attribute(element,2) of
            1 : iren:=iren+1;
            2 : tren:=tren+1;
            3 : bren:=bren+1;
            4 : cren:=cren+1;
          end;
          display_statistics;
          set_attribute(element,2,0);
          move_on(element,queue_set(c,q),decide);
          service_decision(element);
        end;
      end;
    end;
  end;

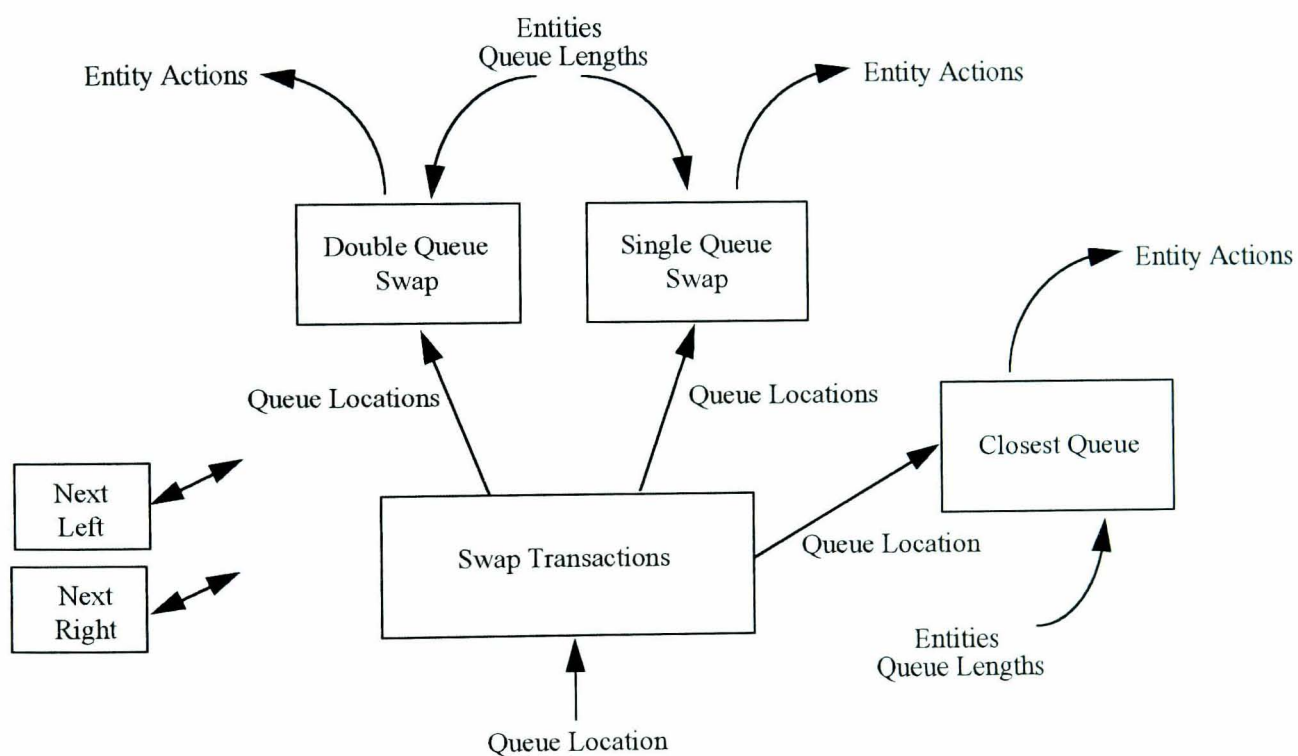
```

Queue Swap Decision

The queue swap decision is the most conventional of the decision modules, containing no neural network components. No swap decision is required for the Information service since it has a single queue. The Business and Currency services have modules with a very similar structure. The Transactions service requires a more complex module due to the larger number of possible queues.

The Business and Currency modules only involve a single procedure each. These are called when the state of particular queue changes (someone is served, or a new counter opens). They determine which customers in the other queue would benefit from swapping queues and check whether those customers do swap. The routines then activate the entities to swap queues.

The Transactions queue swapping follows the same principles as the other, but having more possible queues, the task is more complicated. A single procedure calls a number of others to perform the task. Note that *next_left* and *next_right* are functions that are called by most of the other routines to determine the nearest open queue to the left or right of the queue of interest.



The coding for the queue swap decisions is as follows :

```

{** QUEUE SWAPPING **}

function next_left(p:integer):integer;
{** trnasactions: find nearest open counter to left of the one which the **}
{** customer is currently in **}
var
  i:integer;
  res:integer;
begin
  {** find next open counter to the left **}

```

```

res:=0;
i:=p;
while (i>1) and (res=0) do
  begin
    dec(i);
    if transactions[i]=open then
      res:=i
    end;
  next_left:=res
end;

function next_right(p:integer):integer;
{** trnasactions: find nearest open counter to right of the one which the **}
{** customer is currently in **}
var
  i:integer;
  res:integer;
begin
  {** find next open counter to the right **}
  res:=0;
  i:=p;
  while (i<5) and (res=0) do
    begin
      inc(i);
      if transactions[i]=open then
        res:=i
      end;
    next_right:=res
  end;

function single_trans_swap(q,nextq:integer):boolean;
{** case where only customers from one neighbouring queue would **}
{** benefit from swapping queues **}
{** check if any customers from other queue would benefit from **}
{** swapping **}
{** looks at all customers who would gain, starting from nearest the **}
{** front, and gives a 0.9167 chance that a customer will swap **}
var
  index:integer;
  element:entity;
  swap:boolean;
begin
  swap:=false;
  if size_of(trans_q[q])+size_of(trans_c[q])<size_of(trans_q[nextq]) then
    begin
      index:=size_of(trans_q[q])+size_of(trans_c[q])+1;
      while (size_of(trans_q[q])+size_of(trans_c[q])<size_of(trans_q[nextq]))
        and (index<=size_of(trans_q[nextq])) do
        begin
          if (rnd(27)<0.9167) then {** chance that customer will swap **}
            begin
              element:=identity(index,trans_q[nextq]);
              set_attribute(element,3,q);
              move_on(element,trans_q[nextq],trans_q[q]);
              swap:=true;
            {** note customer behind shuffle forward, so index the same **}

```

```

    end
  else
    inc(index);  {** check next customer **}
  end;
  {** if the server is free and no-one has swapped queues, then **}
  {** the person at the back WILL swap **}
  if (size_of(trans_q[q])+size_of(trans_c[q])=0)
  and (size_of(trans_q[nextq])>0) then
    begin
      element:=identity(size_of(trans_q[nextq]),trans_q[nextq]);
      set_attribute(element,3,q);
      move_on(element,trans_q[nextq],trans_q[q]);
      swap:=true;
    end
  end;
  single_trans_swap:=swap;
end;

function double_trans_swap(q,lq,rq:integer):boolean;
{** case where a queue has two neighbouring queues where customers from **}
{** both could benefit from swapping queues **}
{** there is a probability 0.9167 that an individual customer who **}
{** could gain by swapping queues will do so **}
var
  firstq,secondq:integer;
  qlen,fqlen,slen:integer;
  indexf,indexs:integer;
  element:entity;
  swap:boolean;
begin
  {** decide which queue to consider first **}
  if rnd(31)<0.5 then
    begin
      firstq:=lq;
      secondq:=rq
    end
  else
    begin
      firstq:=rq;
      secondq:=lq
    end;

  swap:=false;
  qlen:=size_of(trans_q[q])+size_of(trans_c[q]);
  fqlen:=size_of(trans_q[firstq])+size_of(trans_c[firstq]);
  slen:=size_of(trans_q[secondq])+size_of(trans_c[secondq]);

  if (qlen<fqlen-1) and (qlen>=slen-1) then {** only first queue benefits **}
    swap:=single_trans_swap(q,firstq) {** treat as 1 queue case **}
  else if (qlen>=fqlen-1) and (qlen<slen-1) then {** only seconds queue benefits **}
    swap:=single_trans_swap(q,secondq) {** treat as 1 queue case **}
  else if (qlen<fqlen-1) and (qlen<slen-1) then
    begin {** two queue case **}
      indexf:=qlen+1;
      indexs:=qlen+1;

      while ((qlen<fqlen-1) or (qlen<slen-1))
      and ((indexf<=fqlen) or (indexs<=slen)) do

```



```

begin
  if (qlen<fqlen-1) and (indexf<=fqlen) then  {** first queue **}
    if (rnd(27)<0.9167) then  {** chance that customer swaps **}
      begin
        element:=identity(indexf,trans_q[firstq]);  {** customer **}
        set_attribute(element,3,q);  {** store new queue location **}
        move_on(element,trans_q[firstq],trans_q[q]);  {** change queues **}
        swap:=true;
        inc(qlen);  {** new queue longer **}
        dec(fqlen);  {** old queue shorter **}
      end
    else {** rnd fails **}
      inc(indexf);  {** next customer in queue **}
    if (qlen<sqlen-1) and (indexs<=sqlen) then  {** second queue **}
      if (rnd(27)<0.9167) then {** chance that customer swaps **}
        begin
          element:=identity(indexs,trans_q[secondq]); {** customer **}
          set_attribute(element,3,q);  {** store new queue **}
          move_on(element,trans_q[secondq],trans_q[q]);  {** change queues **}
          swap:=true;
          inc(qlen);  {** new queue longer **}
          dec(sqlen);  {** old queue shorter **}
        end
      else {** rnd fails **}
        inc(indexs);
      end;
    {** if service for q is free, then if there is anyone waiting **}
    {** in either of the neighbouring queues, move the person from **}
    {** the back - work on basis of the person from the back of the **}
    {** shortest queue, giving firstq priority if they are the same **}
    {** length **}
    if (qlen=0) and (fqlen<=sqlen) then
      begin
        element:=identity(size_of(trans_q[firstq]),trans_q[firstq]);
        set_attribute(element,3,q);
        move_on(element,trans_q[firstq],trans_q[q]);
        swap:=true;
      end
    else if (qlen=0) and (sqlen<fqlen) then
      begin
        element:=identity(size_of(trans_q[secondq]),trans_q[secondq]);
        set_attribute(element,3,q);
        move_on(element,trans_q[secondq],trans_q[q]);
        swap:=true;
      end;
    end;
    double_trans_swap:=swap;
  end;

procedure closest_trans_queue(q:integer);
{** covers rare cases for transaction queues, where there is a free **}
{** server, but no-one queuing in the neighbouring queues **}
{** looks for nearest queuing customers for other counters **}
var
  lq,rq:integer;
  found_left,found_right:boolean;
  queue:integer;

```

```

begin
if size_of(trans_q[q])+size_of(trans_c[q])=0 then    {** if server free **}
begin
{** find nearest queue to left with anyone in it **}
found_left:=false;
lq:=q;
repeat
lq:=next_left(lq);
if lq>0 then
if size_of(trans_q[lq])>0 then
found_left:=true;
until (lq=0) or found_left;
{** find nearest queue to right with anyone in it **}
found_right:=false;
rq:=q;
repeat
rq:=next_right(rq);
if rq>0 then
if size_of(trans_q[rq])>0 then
found_right:=true;
until (rq=0) or found_right;
{** find which valid queue is closest, and move last person to empty queue **}
if (lq=0) and (rq=0) then    {** no other valid queues **}
queue:=0
else if (lq>0) and (rq=0) then    {** only valid queue to left **}
queue:=lq
else if (lq=0) and (rq>0) then    {** only valid queue to right **}
queue:=rq
else if abs(q-lq)<abs(q-rq) then    {** left queue closest **}
queue:=lq
else if abs(q-lq)>abs(q-rq) then    {** right queue closest **}
queue:=rq
else if rnd(31)<0.5 then    {** random choice **}
queue:=lq
else
queue:=rq;

if queue>0 then    {** person exist to go to counter **}
begin
element:=identity(size_of(trans_q[queue]),trans_q[queue]);
set_attribute(element,3,q);
move_on(element,trans_q[queue],trans_q[q]);
end;
end;
end;

procedure swap_transaction(q:integer);
{** determines if any customers in another transactions queue **}
{** can benefit from swapping, and tests if they do swap **}
var
lq,rq,nextlq,nextrq:integer;
swap:boolean;
begin
lq:=next_left(q);
rq:=next_right(q);
swap:=false;

if (lq>0) and (rq=0) then    {** only a queue to the left **}

```



```

    swap:=single_trans_swap(q,lq)
else if (lq=0) and (rq>1) then
    swap:=single_trans_swap(q,rq)
else if (lq>0) and (rq>0) then
    swap:=double_trans_swap(q,lq,rq);

if swap then    {** check for shuffle effect from surrounding queue **}
begin
    while lq>1 do    {** check queues to the left **}
    begin
        nextlq:=next_left(lq);    {** find next queue to left **}
        if nextlq>0 then    {** if one exists then test for swaps **}
        begin
            swap:=single_trans_swap(lq,nextlq);
            if swap then
                lq:=nextlq    {** if swap occurs then go on to next queue *}
            else    {** no swap **}
                lq:=0    {** otherwise stop **}
            end
        else    {** no queue to left **}
            lq:=0    {** if no more queues to left then stop **}
        end;
    while rq>1 do    {** check queues to the right **}
    begin
        nextrq:=next_right(rq);    {** find next queue to right **}
        if nextrq>0 then    {** if one exists then test for swaps **}
        begin
            swap:=single_trans_swap(rq,nextrq);
            if swap then
                rq:=nextrq    {** if swap occurs then go on to next queue *}
            else    {** no swap **}
                rq:=0    {** otherwise stop **}
            end
        else    {** no queue to right **}
            rq:=0    {** if no more queues to right then stop **}
        end;
    end;
end;

{** deal with cases where server is free, customer from back of nearest **}
{** queue can join **}
closest_trans_queue(q);
end;

procedure swap_business(q:integer);
{** determines if any customers in the other business queue **}
{** can benefit from swapping, and tests if they do swap **}
var
    index:integer;
    element:entity;
begin
    {** check if any customers from other queue would benefit from **}
    {** swapping : note 3-q gives index of other queue **}
    {** if other queue is shut, then size is 0 so no swaps would happen **}
    {** looks at all customers who would gain, starting from nearest the **}
    {** front, and gives a 0.9167 chance that a customer will swap **}
    if size_of(bus_q[q])+size_of(bus_c[q])<size_of(bus_q[3-q]) then
        begin

```

```

index:=size_of(bus_q[q])+size_of(bus_c[q])+1;
while (size_of(bus_q[q])+size_of(bus_c[q])<size_of(bus_q[3-q]))
and (index<=size_of(bus_q[3-q])) do
  begin
    if (rnd(29)<0.9167) then    {** chance that customer will swap **}
      begin
        element:=identity(index,bus_q[3-q]);
        set_attribute(element,3,q);
        move_on(element,bus_q[3-q],bus_q[q]);
        {** note customer behind shuffle forward, so index the same **}
      end
    else
      inc(index); {** check next customer **}
    end;
  {** if the server is free and no-one has swapped queues, then **}
  {** the person at the back WILL swap **}
  if (size_of(bus_q[q])+size_of(bus_c[q])=0)
  and (size_of(bus_q[3-q])>0) then
    begin
      element:=identity(size_of(bus_q[3-q]),bus_q[3-q]);
      set_attribute(element,3,q);
      move_on(element,bus_q[3-q],bus_q[q]);
    end
  end
end;

```

```

procedure swap_currency(q:integer);
{** determines if any customers in the other currency queue **}
{** can benefit from swapping, and tests if they do swap **}
var
  index:integer;
  element:entity;
begin
  {** check if any customers from other queue would benefit from **}
  {** swapping : note 3-q gives index of other queue **}
  {** if other queue is shut, then size is 0 so no swaps would happen **}
  {** looks at all customers who would gain, starting from nearest the **}
  {** front, and gives a 0.9167 chance that a customer will swap **}
  if size_of(curr_q[q])+size_of(curr_c[q])<size_of(curr_q[3-q]) then
    begin
      index:=size_of(curr_q[q])+size_of(curr_c[q])+1;
      while (size_of(curr_q[q])+size_of(curr_c[q])<size_of(curr_q[3-q]))
      and (index<=size_of(curr_q[3-q])) do
        begin
          if (rnd(30)<0.9167) then    {** chance that customer will swap **}
            begin
              element:=identity(index,curr_q[3-q]);
              set_attribute(element,3,q);
              move_on(element,curr_q[3-q],curr_q[q]);
              {** note customer behind shuffle forward, so index the same **}
            end
          else
            inc(index); {** check next customer **}
          end;
        {** if the server is free and no-one has swapped queues, then **}
        {** the person at the back WILL swap **}
        if (size_of(curr_q[q])+size_of(curr_c[q])=0)

```

```
and (size_of(curr_q[3-q])>0) then
  begin
    element:=identity(size_of(curr_q[3-q]),curr_q[3-q]);
    set_attribute(element,3,q);
    move_on(element,curr_q[3-q],curr_q[q]);
  end
end
end;
```